POLITECNICO DI TORINO
Repository ISTITUZIONALE

Runtime Feature Compression for Adaptive Keyword Spotting on Embedded Systems

Original
Runtime Feature Compression for Adaptive Keyword Spotting on Embedded Systems / Peluso, V., Calimera, A., Macii, E., Montuschi, P.. - In: IEEE INTERNET OF THINGS JOURNAL. - ISSN 2327-4662. - 13:11(2026), pp. 25286-25301. [10.1109/jiot.2026.3676917]

Availability:
This version is available at: 11583/3010606 since: 2026-05-06T12:55:26Z

Publisher:
Institute of Electrical and Electronics Engineers

Published
DOI:10.1109/jiot.2026.3676917

Terms of use:

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

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

11 November 2026

Runtime Feature Compression for Adaptive Keyword Spotting on Embedded Systems

Valentino Peluso *Member, IEEE*, Andrea Calimera *Member, IEEE*, Enrico Macii *Fellow, IEEE*, Paolo Montuschi *Fellow, IEEE*

***Abstract*—Voice user interfaces rely on keyword spotting (KWS) to detect wake-word commands, enabling low-power devices to switch from drowsy to active states and initiate more complex tasks. In embedded systems, KWS combines handcrafted acoustic features extraction with lightweight neural network classifiers to achieve accurate detection within strict resource constraints. Adapting KWS to time-varying energy budgets requires optimization strategies that operate at runtime. Most existing approaches adjust the complexity of the neural model but overlook that a substantial amount of latency, and thus energy consumption, is due to feature extraction, which remains unaffected by model scaling. This work introduces *Runtime Feature Compression* (RFC), a dynamic rescaling strategy that modulates the workload of the entire KWS pipeline. RFC promotes the *hop-length* parameter of the Short-Time Fourier Transform as a runtime control knob to adjust the number of time frames in speech features, allowing a single model to operate across multiple latency modes. To support this flexibility, we introduce two training-time techniques: *HopAugment*, a data augmentation scheme that exposes the model to variable hop lengths during training, and *Masked Layers*, which preserve consistent activation statistics during training and inference under compressed feature settings. Evaluations on four KWS datasets using the TC-ResNet model family show that RFC outperforms model scaling techniques, offering a wider range of latency-accuracy trade-offs. RFC achieves up to 31.8% lower latency without accuracy degradation, or up to 0.30% higher accuracy within equivalent latency bounds. That proves RFC improves adaptability in energy-constrained IoT speech interfaces. A set of ablation studies further demonstrates the robustness of RFC by evaluating the role of its training components, batching strategies, ability to preserve accuracy with a shared weight set, scalability across operating modes, and applicability to different model architectures.**

***Index Terms*—Embedded Systems, Internet of Things, Keyword Spotting, Adaptive Neural Networks, Runtime Reconfigurable Inference.**

I. Introduction

THE growing popularity of consumer voice assistants in smart speakers and smartphones has significantly reshaped expectations for voice-based interactions in portable and wearable electronics. Intuitive conversational interfaces are now considered essential across a broad range of devices, including laptops [1], extended reality headsets [2], and hearing aids [3], many of which are battery-powered and operate largely in low-power mode to preserve energy [4]. Keyword Spotting (KWS), the task of detecting predefined single-word voice commands, is commonly used as the primary mechanism to wake the devices from drowsy to active power modes. One common application is using KWS as the entry point for smart assistants, where a detected wake word activates the system and triggers the transmission of more complex user queries to cloud-based speech processing services. KWS systems must be fast to enable natural user interaction and deliver responsive experiences. At the same time, they must be accurate to prevent unintended activations that could trigger unnecessary audio transmission to the cloud, thereby wasting communication energy and potentially exposing confidential conversations. These issues directly relate to the dual challenges of energy efficiency and privacy that are increasingly critical in next-generation communication networks [5]–[7]. The ultimate goal is to design optimized and compact neural inference pipelines that deliver low-latency and energy-efficient processing, while reducing dependence on cloud-based computation [8]–[10].

Prior works demonstrated the feasibility of resource-constrained KWS systems [11]–[19] using a two-stage software pipeline consisting of a signal-processing front-end (FE) followed by a neural network (NN) classifier. The front-end implements the feature extraction computing log-Mel spectrograms [16], [19] or Mel-Frequency Cepstral Coefficients (MFCCs) [11]–[15], [17], [18], which are then fed to the neural network to estimate the likelihood that the audio stream contains a target command. Although more sophisticated alternatives exist, the FE+NN architecture is well-suited for tiny IoT devices [20], [21] as it offers an effective trade-off between computational efficiency and accuracy.

A current open issue is the adaptability of embedded KWS systems to a wide range of hardware configurations and operating conditions. Distributed devices deployed in the field may differ in terms of hardware specifications or energy budget. Even on the same device, performance and compute capability change over time due to factors such as residual battery life or thermal conditions. Therefore, the same, statically optimized KWS pipeline may result in degraded responsiveness or lower prediction quality under adverse conditions. A high efficiency requires adaptive strategies that dynamically adjust the workload based on runtime resource availability and constraints, without compromising the quality of predictions.

A naive, yet practical approach to cope with hardware

Valentino Peluso, Andrea Calimera, and Paolo Montuschi are with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy (e-mail: valentino.peluso@polito.it; andrea.calimera@polito.it; paolo.montuschi@polito.it).

Enrico Macii is with the Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino, 10129 Turin, Italy (e-mail: enrico.macii@polito.it).

Copyright © 2026 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

diversity is to train a family of neural network models that share the same architecture backbone but differ in information capacity and computational complexity, for instance, using a different depth (number of layers) and width (number of filters per layer) [11]–[19]. Each neural network has a different weight set, and the most suitable variant is deployed based on the hardware constraints and specifications. This method is less efficient for runtime adaptation due to substantial memory overhead and power and performance penalty for storing and loading multiple weight sets on the same device. A more flexible option consists of adaptive neural networks that can rescale at runtime. Prior works explored different dynamic strategies that adjust model depth [22]–[30], width [31]–[35], precision [36]–[38], or compound attributes [39], based on external triggers like the available energy, the input complexity, or the desired accuracy. When applied to KWS pipelines, these methods share a common limitation: only the neural model complexity is rescaled, leaving the front-end stage untouched. This is a source of weakness as the front-end (i) dominates the total latency and energy consumption, and (ii) it defines the size of features, which affects downstream compute and memory loads. Ignoring these factors leads to inefficient adaptive solutions.

This work introduces *Runtime Feature Compression* (RFC), a novel online rescaling strategy that dynamically adjusts the computational complexity of the entire KWS pipeline, and hence the total latency and energy consumption. RFC operates the *hop-length* parameter available in the Short-Time Fourier Transform (STFT) of the front-end as a runtime control knob. Unlike prior adaptive approaches that rely on neural-network topology scaling through depth, width, or precision modulation, RFC exploits hop-length rescaling as an orthogonal runtime knob to concurrently reduce the computational cost of the front-end and lower the complexity of the neural inference by compressing the temporal cardinality of the input features. Modulating the hop length changes the number of time frames in the extracted features, thereby lightening the CPU workload for both the front-end and the neural network. Features extracted with a smaller hop length show a finer temporal granularity and yield better accuracy. A larger hop length coarsens the temporal granularity, lowering the number of arithmetic and memory instructions needed and bringing substantial latency and energy savings at the cost of marginal quality degradation. By working on the front-end stage, RFC enables dynamic switching among multiple operating points without modifying the neural network architecture or storing entire model families. To make RFC feasible and attainable, this work also presents two training-time techniques, *HopAugment* and *Masked Layers*, whose aim is to maximize model accuracy under different front-end configurations, enhancing the quality-performance trade-off over the operating range. RFC is architecture-agnostic and can be applied to existing neural models without structural changes.

We tested the RFC method using a KWS pipeline built on TC-ResNet [12], a neural model architecture optimized for deployment on resource-constrained embedded systems. Training and evaluation were performed using four publicly available KWS datasets. Experimental results demonstrate that RFC offers a broader performance dynamic range, outperforming model-only scaling strategies. Specifically, RFC achieves up to 31.8% lower latency without accuracy loss, or up to 0.30% higher accuracy for the same latency. Moreover, the experiments validated the enhanced training flow customized for RFC, as models trained without it show substantial accuracy drops, up to 15.17% in the worst case. These results confirm that RFC substantially enhances the adaptability, efficiency, and robustness of KWS in resource-constrained settings that are typical of IoT applications. A comprehensive set of ablation studies ultimately proves the robustness of RFC by isolating the impact of the proposed training techniques, analyzing generalization across hop-length configurations under different batching strategies, comparing against hop-specific specialized models, evaluating scalability across multiple runtime operating points, and assessing applicability to alternative neural architectures.

The main contributions of this work are as follows:

- propose a runtime adaptation mechanism that jointly modulates front-end feature extraction and neural inference through hop-length scaling;
- introduce training-time techniques (HopAugment and Masked Layers) that enable accurate and stable inference under variable feature granularity;
- demonstrate that a single model trained and operated with RFC can efficiently support a broad spectrum of accuracy-latency operating points, consistently outperforming approaches based on neural-network topology scaling.

The remainder of this paper is organized as follows. Section II reviews the KWS pipeline and motivates our approach. Section III summarizes recent advancements in runtime-adaptive inference. Section IV presents the RFC framework and the proposed training-time techniques. Section V details the experimental setup, while Section VI shows the results and their analysis. Finally, Section VII concludes the paper.

II. Background on Embedded KWS Pipelines

The typical workflow for embedded KWS consists of a two-stage pipeline: a front-end (FE) for feature extraction and a neural network (NN) for classification. The FE transforms the raw input speech signal into compact representations, such as log-Mel spectrograms or MFCCs; the NN processes such features to identify the most likely keyword.

A. Front-end

The first substage of the FE is the Short-Time Fourier Transform (STFT). The STFT segments the raw speech waveform into partially overlapped frames of fixed length defined by the *window-length* parameter. Each frame is multiplied by a Hann function such that the signal tapers on the boundaries and the spectral leakage is reduced before the Discrete Fourier Transform (DFT). The DFT is computed over a fixed number of frequency bins, commonly referred to as the *FFT size*, selected to be greater than or equal to the window length, and constrained to a power of two in order to enable efficient Fast Fourier Transform (FFT) computation. When the FFT size

exceeds the window length, the input frame is zero-padded before transformation, increasing frequency-domain sampling density without altering the underlying spectral content.

The resulting frequency spectrum is converted into a spectrogram by calculating the squared magnitude of the complex values. Smaller values of window length enhance the temporal resolution of the STFT, improving the ability to distinguish events that are close in time, but reduce the frequency resolution, thereby lowering the ability to distinguish closely spaced tones. The opposite does hold: larger values of the window length reduce the temporal resolution in favor of higher frequency resolution. This trade-off, known as the *time-frequency localization trade-off*, must be balanced based on the characteristics of the input signal. In speech processing, the window length is typically in the range 20–40 ms [40], [41], as speech is considered stationary over short timescales. The *hop-length* parameter defines the stride between consecutive frames, controlling the overlaps. Smaller hop lengths help compensate for the information loss introduced by the Hann function, which attenuates the signal near the window boundaries, ensuring smoother temporal coverage between adjacent frames. However, smaller hop lengths result in a higher number of frames, and thus more FFT computations, increasing the computational cost.

The spectrogram is further processed using a set of triangular filter banks, known as Mel filters, spaced according to the Mel scale. The Mel scale is a nonlinear frequency scale that aligns with human auditory perception to emphasize the lower frequencies. Moreover, Mel filters aggregate multiple frequency bands into a single Mel frequency, reducing feature dimensionality. This operation is implemented as a matrix multiplication between the spectrogram frames and a coefficient matrix representing the Mel filter bank. The output is passed through a logarithmic function, producing the log-Mel spectrogram, which mimics the human auditory perception of loudness following a logarithmic scale.

The log-Mel spectrogram can be fed as an input feature to the NN [16], [19] or further transformed via the Discrete Cosine Transform (DCT) to compute MFCC features [11]–[15], [17], [18]. The latter are usually preferred as they better catch the spectral envelope of the speech signal and are especially effective in phoneme-level recognition. Like for Mel filtering, the DCT is implemented as a matrix multiplication between the log-Mel frames and a set of DCT coefficients. Given their popularity in embedded KWS implementation, in this work we also opted for the MFCCs. However, our method is compatible with both types of features.

B. Neural Networks

Most NNs deployed in embedded KWS pipelines are Convolutional Neural Networks (CNNs), which have proven to reach the best trade-off between accuracy and computational cost. The proposed CNN architectures can be broadly characterized based on the dimensionality of the convolutional kernels. There are models like [11], [15] that run 2D convolutions across both time and frequency dimensions of the input features, and models that leverage 1D convolutions, often called temporal convolutions [12], [14], [17] or time-channel convolutions [13], where filters slide along the time axis and integrate over frequency. A key advantage of 1D convolutions is their ability to catch the whole frequency spectrum, whereas 2D convolutions are limited to local receptive fields that cover a few consecutive bands at a time due to the limited kernel size. A hybrid strategy named broadcasted residual learning [16] combines both 1D and 2D convolutions to join the strengths of each approach. Recent works [18], [19] also explored cumbersome Transformer-based architectures.

Beyond the choice of the NN template (2D, 1D, a mix of the two, or Transformers), an orthogonal design and implementation issue concerns the scalability of the model to different hardware settings and specifications. The mainstream approach is to build and train a family of models with optimal accuracy-efficiency trade-offs, leaving developers free to select the most suited version at compile time, according to the hardware constraints. Variants within the same family are generated through *topology scaling*, i.e., by rescaling the network topology, both in width, by adjusting the number of channels per layer, and depth, by tuning the number of layers. The resulting model variants are typically trained using the same FE configurations, despite there is evidence that joint optimization of the FE hyper-parameters and NN topology can yield superior results [40], [41], [43], [44]. It is worth emphasizing that each variant must be trained independently, resulting in substantial design and implementation overhead. Moreover, this model-family paradigm barely meets the needs of runtime scaling. Storing multiple models on-device and loading the most appropriate one at runtime based on contextual variables, such as input complexity [45], processing load [46], [47], or on-chip temperature [48], has several drawbacks.

To better address the challenges posed by hardware heterogeneity and runtime adaptability, recent research has explored more flexible and efficient adaptive strategies where a single model is trained to operate across multiple configurations, each representing a different point in the accuracy-efficiency spectrum, while sharing a common set of weights. The following section reviews the most promising of these approaches, introducing a novel taxonomy to organize and compare them.

III. Previous Work on Adaptive Neural Networks

Despite the growing interest in adaptive neural networks, relatively few techniques have been explicitly developed for KWS. In this section, we thereby broaden the review to include approaches from other domains such as image classification (IC), depth estimation (DE), and object detection (OD), which operate in embedded scenarios under similar resource constraints. Table I summarizes a representative subset of those methods. For a more comprehensive survey, we refer the reader to [49]. Our taxonomy is organized along four dimensions: (i) the control knob employed, (ii) the external trigger prompting the adaptive scheme, (iii) the pipeline module affected by the adaptive scheme (i.e., front-end, neural network), and (iv) the application domain for which the method was originally developed and evaluated.

TABLE I
COMPARISON OF DIFFERENT IMPLEMENTATIONS OF ADAPTIVE NEURAL NETWORKS BASED ON THE CONTROL KNOB, THE TRIGGERING CONDITION, THE AFFECTED PROCESSING MODULE (FE: FRONT-END, NN: NEURAL NETWORK), AND THE TARGET TASK (IC: IMAGE CLASSIFICATION, OD: OBJECT DETECTION, DE: DEPTH ESTIMATION, KWS: KEYWORD SPOTTING).

Method	Control Knob	Triggering Condition	Module	Task
BranchyNet [22]	Model Depth	Input Complexity	NN	IC
Runtime Configurable DNNs [31]	Model Width	Energy, Input Complexity	NN	IC
Runtime Neural Pruning [32]	Model Width	Input Complexity	NN	IC
ConvNet with AIGs [23]	Model Depth	Input Complexity	NN	IC
SkipNet [24]	Model Depth	Input Complexity	NN	IC
Dynamic Bit-width Reconfiguration [36]	Model Precision	Input Complexity	NN	IC
Feature Boosting and Suppression [33]	Model Width	Input Complexity	NN	IC
Slimmable Neural Networks [34]	Model Width	Energy	NN	IC
Scalable-Effort ConvNets [37]	Model Precision	Abstraction Level	NN	IC
Energy-Driven Precision Scaling [38]	Model Precision	Energy	NN	IC
Multi-Exit Neural Networks [25]	Model Depth	Energy, Input Complexity	NN	IC
EQPyD-Net [26]	Model Depth	Energy	NN	DE
WeNet [35]	Model Width	Energy	NN	IC
EdgeCompress [39]	Model Depth & Width	Input Complexity	NN	IC
Gated Compression Layers [27]	Model Depth	Input Complexity	NN	IC, KWS
T-RecX [28]	Model Depth	Input Complexity	NN	IC, KWS
LoCoExNet [29]	Model Depth	Input Complexity	NN	IC
VALO [42]	Input Regions-of-Interest	Input Complexity	NN	OD
Temporal Early Exit Networks [30]	Model Depth	Input Complexity	NN	IC, KWS
This work	Hop Length	Energy	FE+NN	KWS

A. Control knobs

The majority of control knobs derive from static optimization strategies such as pruning and quantization. Both exploit the over-parameterization of neural networks but act on different structural characteristics of the model. Pruning reduces model complexity by removing components that contribute minimally to prediction accuracy. For instance, methods based on adaptive *model width* dynamically disable entire convolutional channels at runtime, guided by pre-trained agents or applying global scaling factors (e.g., width multipliers) [31]–[35]. Similarly, *model depth* adaptations operate at a coarser level, selectively bypassing entire layers to reduce computation [22]–[30]. These two approaches are orthogonal and can be combined for compound scaling [39]. Quantization leverages the intrinsic resilience of neural networks to arithmetic precision by lowering the bit width of weights and activations to improve efficiency. Adaptive *model precision* techniques dynamically adjust the arithmetic precision at runtime, often at the granularity of individual layers for the sake of regularity and scalability [36]–[38].

Custom control knobs can be tailored for a specific task. For example, VALO [42] proposed an anytime object detection framework that exploits spatial information to selectively identify regions-of-interest to be filtered and processed first, bypassing less relevant portions of the input and eventually enabling progressive refinements over time of the prediction until resources are available.

To our knowledge, no runtime control knob has been proposed that specifically targets KWS. This gap motivates our work, which introduces the *hop length* of the STFT as a KWS-specific dynamic knob.

B. Triggering Conditions

Another differentiating aspect among adaptive methods is the external condition that triggers the scaling mechanism. Common triggers include *abstraction level*, *input complexity*, and *energy budget*.

a) Abstraction Level: Neural networks trained on semantically structured datasets can be adapted to perform classification at different levels of granularity. As shown in [37], the abstraction level may serve as an external trigger that governs computational effort. Coarse classification (e.g., distinguishing animals from vehicles) requires less computation than fine-grained classification (e.g., identifying species or models). Efficient systems exploit these semantic relationships to adapt the computational load to contextual needs and use a specific control knob (e.g., arithmetic precision [37]) to minimize resource usage accordingly. While this approach is effective in large-scale classification problems with thousands of hierarchically structured classes, it is less relevant for KWS, where the number of target classes in the dataset is limited and lacks semantic organization.

b) Input Complexity: A widely adopted triggering mechanism is the input complexity, which assumes that not all inputs require equal computational effort. Certain patterns are easier to classify due to salient feature exposure, while others are more ambiguous and require more effort [50]. In response, several methods apply iterative refinement by gradually increasing computational effort until a confident prediction is reached. This may involve activating additional layers [22]–[25], [27]–[30] or channels [31]–[33], or increasing arithmetic precision [36] till a consensus is reached. Statistically, as most of the inputs have lower complexity, the overall effort and resources consumed are lower. Being data-driven, these strategies lack direct control as the trigger is embedded into the input itself. Indeed, they suffer from unpredictability and performance variance, with potential risk of energy overhead and penalties [36], which is a key limiting factor for applications with real-time constraints.

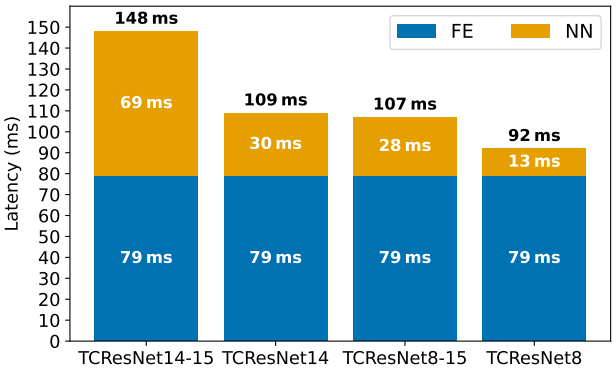


Fig. 1. Front-end (FE) and neural network (NN) latency breakdown in KWS pipelines built with the TC-ResNet model family on an STM32F767ZI MCU.

c) Energy: A more deterministic control is achieved by using the energy budget as a direct constraint to drive the control knob. Several neural network configurations are defined and optimized at training time (e.g., via pruning [31], [34], [35] or quantization [38]), then stored on chip and deployed at runtime according to the remaining energy. This avoids the unpredictability of data-driven triggers and enables reliable, low-power operation, which is paramount for real-time applications like KWS.

C. Target Modules & Tasks

The existing methods listed in Table I share a common limitation: they adopt a model-centric view. Indeed, they apply a predefined scaling mechanism operating on the neural network, while the front-end stage remains unaffected. This is mainly due to their origins in computer vision tasks, and image classification in particular, where the pre-processing stage is minimal, and the neural inference dominates the overall computational effort. The few adaptive KWS methods that do exist inherit this characteristic and share the same model-centric view, for instance, proposing layer pruning as the control knob [27], [28], [30]. However, in embedded KWS pipelines, the front-end represents a dominant portion of the total workload, and rescaling the neural network alone offers marginal benefits in terms of latency and energy consumption.

For a quantitative analysis, we profiled end-to-end latency for different topology configurations of the TC-ResNet family [12], a standard suite for embedded KWS. The evaluated models include TC-ResNet8, TC-ResNet8-1.5, TC-ResNet14, and TC-ResNet14-1.5; the numeric suffix denotes the network depth, namely, the number of layers, and the 1.5 factor denotes the width multiplier applied to each layer to enlarge the number of convolutional filters for a higher accuracy [12]. Figure 1 presents latency breakdowns measured on a STM32F767ZI ARM Cortex-M7 microcontroller unit (MCU) (see Section V for a detailed description of the experimental setup), highlighting the relative contributions of the front-end stage (blue) and the neural network (orange). In all cases, the front-end dominates execution time, ranging from 53.4% (TC-ResNet14-1.5) to 85.9% (TC-ResNet8), and the overall latency improvement achievable via topology scaling is limited to 1.6×. Intuitively, model scaling leads to limited latency (hence energy) savings, emphasizing the need for strategies that seamlessly cover both the front-end and the neural network. Although our analysis is focused on the TC-ResNet family statically rescaled using different topology settings, the same conclusions hold for other models and dynamic knobs: rescaling the neural network stage alone weakly improves end-to-end performance. This insight motivates the proposed *Runtime Feature Compression* method, which introduces the concept of an adaptive front-end.

D. Beyond State-of-the-Art

The comparative review in Table I highlights three distinguishing aspects of our work. First, we introduce a novel *runtime* control knob that enables dynamic inference by primarily operating on the front-end stage. Second, this control knob allows reshaping the computational complexity of the entire KWS pipeline, jointly affecting both front-end feature extraction and neural network inference. Third, the proposed approach is specifically designed for the KWS domain, rather than being adapted from techniques originally developed for other application areas, such as computer vision, where domain mismatch can limit effectiveness.

Although other prior studies [40], [41], [43], [44] investigated front-end hyperparameter tuning, including optimal hop-length selection for better accuracy-latency trade-off, those methods rely on *static* configurations and require training and deploying separate models for each operating mode. The resulting need to deploy multiple weight sets introduces non-negligible memory overhead on the end devices and seriously limits runtime flexibility in resource-constrained environments. RFC fosters effective runtime adaptation by allowing a single shared weight set to work reliably across multiple hop-length configurations, thereby covering a wide range of accuracy-latency operating points within a unified pipeline. This capability is enabled and made robust through two dedicated training-time techniques (see Section IV) that ensure higher generalization, an objective that cannot be achieved using conventional KWS training pipelines (see Section VI-C).

IV. Runtime Feature Compression

A. Overview

The KWS pipeline, made up of the front-end (FE) processing stage followed by a neural network (NN) classifier, is fed with an input audio waveform $\mathrm{x} \in \mathbb{R}^N$, where N is the number of discrete samples acquired over a fixed temporal window (e.g., $N = 16,000$ for a one-second recording at 16 kHz). The FE stage transforms the 1D waveform into a 2D time-frequency representation X, either a log-Mel spectrogram or Mel-Frequency Cepstral Coefficients (MFCCs), via a transformation $\mathcal{T}_{w,h} : \mathbb{R}^N \rightarrow \mathbb{R}^{F \times T}$, where F is the number of frequency features (Mel bins or cepstral coefficients) and T is the number of time frames. The transformation $\mathcal{T}_{w,h}$ is parametrized by the window length w and hop length h of the STFT. While in a conventional KWS pipeline, h is statically defined at training time and kept fixed during inference, our proposal operates with a variable parameter. Since the number of time frames T scales inversely with

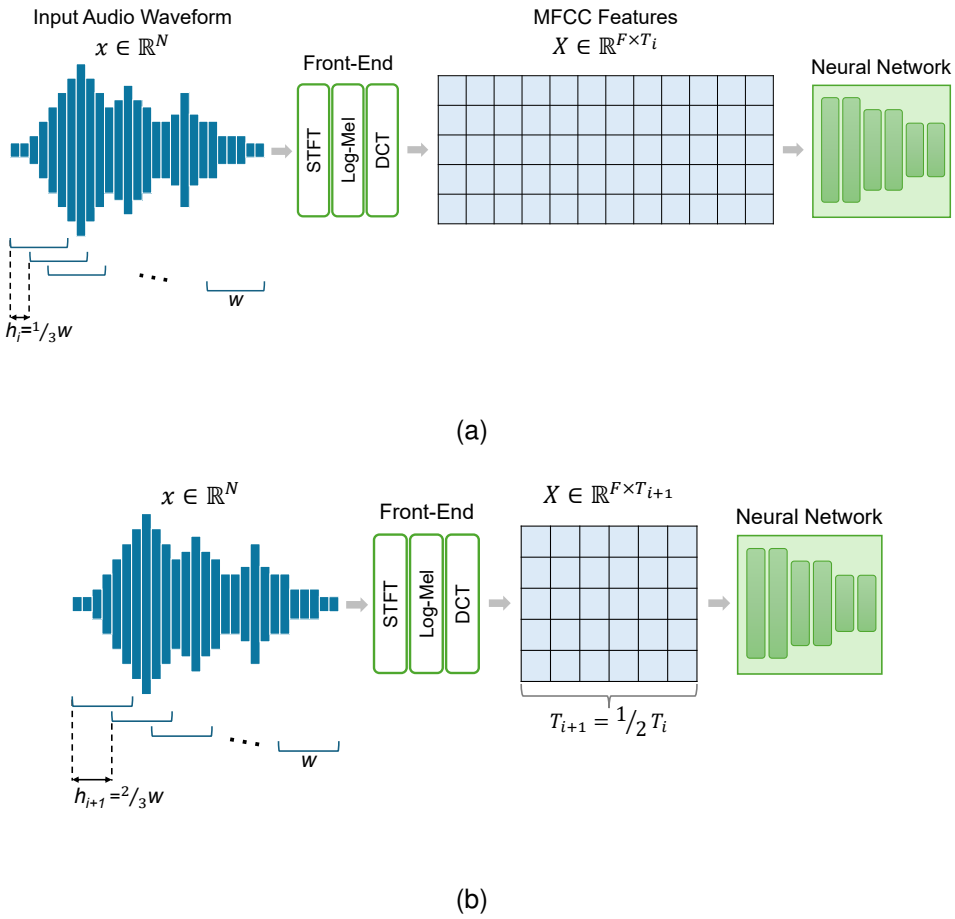


Fig. 2. Schematic overview of the proposed Runtime Feature Compression (RFC) mechanism. RFC introduces the STFT hop length h as a runtime control knob to modulate the temporal dimension of audio features, and hence the computational complexity of both the front-end and the classification stages.

h (i.e., $T \propto 1/h$), modifying h directly affects the computational load across both FE and NN stages. A smaller h increases the overlap between adjacent STFT windows, thereby reducing the information loss due to Hann windowing and enhancing the resulting classification accuracy of the NN stage. A larger h results in fewer frames T, leading to a coarser temporal granularity and hence potential accuracy drops. From a resource perspective, the larger the hop length, the lower the computational costs. Indeed, h indirectly affects the number of FFTs performed, as well as the downstream transformations such as log-Mel filtering and MFCCs, which involve matrix multiplications whose cost scales with the input feature size. Furthermore, the dimensionality of the tensor input to the NN classifier is likewise affected, impacting the number of operations and the latency of the inference stage. Overall, increasing (reducing) h results in faster (slower) and less computationally intensive (more expensive) KWS, with a potential penalty (gain) in accuracy. The proposed RFC mechanism leverages these relationships to implement dynamic energy-quality scaling at runtime. Specifically, a KWS system enhanced with RFC supports a predefined discrete set of hop lengths $\mathcal{H} = \{h_i \mid i \in [1, L], h_i < h_{i+1}\}$, corresponding to L distinct operating modes. Each mode reflects a different trade-off between accuracy and computational cost, allowing on-the-fly adaptation to runtime constraints such as energy availability, thermal conditions, or processor workload.

Figure 2 illustrates the RFC mechanism. It graphically shows how changing the hop length rescales the temporal resolution of extracted features and, consequently, the overall computational burden. The figure depicts two runtime configurations: one with $h_i = \frac{1}{3}w$ (Fig. 2a), resulting in T_i frames, and another with $h_{i+1} = \frac{2}{3}w$ (Fig. 2b), yielding $T_{i+1} = \frac{1}{2}T_i$ frames. The latter configuration, with reduced frame overlap, has reduced representation and higher energy efficiency.

The key advantage of RFC lies in its ability to modulate the computational cost of the entire pipeline without any architectural changes to the deployed model or multiple model instantiations. However, this flexibility introduces a major challenge: models trained under a given hop length experience substantial accuracy degradation once deployed on-field when features are extracted with other hop lengths (see Section VI-C for a quantitative assessment). This issue must be carefully addressed to avoid prediction quality from falling below an acceptable threshold. We thereby introduce two complementary techniques aimed at improving model robustness against variable hop lengths: *HopAugment*, a training-time data augmentation scheme, and *Masked Layers*, a mechanism to fix the batch normalization and global average pooling layers of the NN, ensuring statistical consistency in intermediate features during training and inference.

B. Rationale for Hop-Length-Based RFC

Among the front-end configuration parameters, the hop length is uniquely suited to serve as an effective runtime control knob for latency and energy scaling. Other front-end hyper-parameters provide limited benefits or introduce prohibitive overheads when adjusted at runtime.

Reducing the sampling rate requires band-limited interpolation and anti-aliasing filtering of the input signal. These operations introduce non-negligible computational overhead that can partially or fully offset the savings achieved through low-rate processing, especially in tightly constrained embedded scenarios. Moreover, supporting dynamic resampling typically needs different interpolation kernels for each target rate, introducing extra memory overhead.

Modifying the window length has a marginal impact on computational complexity. For an input waveform of N samples, the number of time frames T generated by the STFT is given by

$$T = \left\lfloor \frac{N - w}{h} \right\rfloor + 1,$$

where w is the window length and h is the hop length. Since $w \ll N$ (the window length is constrained to a narrow range of tens of milliseconds to preserve frequency resolution), any modulation applied to it has a negligible impact on T. Whereas, increasing h proportionally reduces T, with a direct impact on the number of STFT evaluations and feature transformations.

Adjusting the FFT size is similarly unsuited for runtime adaptation. The FFT size is selected as the smallest power-of-two value greater than or equal to w to maximize computational efficiency, leaving no room for additional gains.

Reducing the number of Mel filterbank channels or MFCC coefficients affects the frequency dimension of the features, but not the number of time frames. This severely limits the latency-scaling range achievable by modern embedded KWS models based on temporal convolutions [12], [13], for which the number of time frames defines the number of convolutional windows, and hence the number of operations of the whole network. On the other hand, the frequency dimension influences the computational cost of the first convolutional layer, with no impact on the next layers. As a result, frequency-resolution scaling yields negligible savings. Furthermore, supporting a variable number of Mel bins or MFCC coefficients would require storing separate Mel filterbanks or DCT matrices, increasing the on-chip memory consumption.

Based on the above observations, hop-length scaling is the only FE control knob that can effectively reduce the computational cost of both the feature extraction and neural inference stages, enabling efficient runtime adaptation without introducing additional compute or memory overhead.

C. HopAugment

To enable accurate classification under variable front-end configurations, *HopAugment* implements a training-time data augmentation method that exposes the NN to features extracted with hop lengths spanning the predefined discrete set $\mathcal{H}$. During training, each raw input waveform is randomly assigned a hop length h_i uniformly picked from the discrete set $\mathcal{H}$ and then transformed by the FE to produce the corresponding feature tensor. The procedure helps the model to generalize across varying temporal dimensions introduced by different hop settings, which is paramount for effective runtime adaptation. However, with changes in the hop length, the resulting feature tensors differ in the number of time frames, with shorter hop lengths producing longer sequences. Since deep learning frameworks typically require uniform tensor dimensions within a training batch, all feature tensors are zero-padded along the time axis to match the maximum length observed, i.e., the length corresponding to the smallest hop length in the set $\mathcal{H}$.

It is worth noticing that HopAugment is fully compatible with standard data augmentation techniques employed in KWS training, including random time shifts, additive background noise, and spectro-temporal masking strategies such as SpecAugment [48], which further enhance the model's robustness to temporal jitter, noise conditions, and spectro-temporal masking. As discussed in Section V, we implemented an augmentation pipeline that combines many of such techniques, demonstrating that models trained with HopAugment maintain high accuracy across all hop lengths in $\mathcal{H}$, consistently outperforming fixed-hop baselines.

As a final remark, we emphasize that training-time augmentation specifically designed for variable hop lengths represents a key novelty of this work. Conventional augmentation techniques deployed in image or audio tasks that rely on resizing or resampling incur additional interpolation costs and are therefore unsuitable for adaptive systems aimed at reducing the computational loads. With hop-length modulation, entire portions of the input signal are skipped during STFT computation, resulting in a true reduction of the processing workload. HopAugment enables the neural network to exploit these benefits, preserving accuracy across multiple hop-length configurations while fully realizing the computational gain.

D. Masked Layers

The CNNs deployed in KWS pipelines typically include Batch Normalization (BN) layers to improve training stability and Global Average Pooling (GAP) layers to reduce the size of intermediate features. Unfortunately, the output of these two layers is affected by the zero-padding introduced by HopAugment on the input features to unify temporal dimensions across the batch. This padding skews the activation statistics computed by BN and GAP, leading to distribution mismatches between training (with padding) and inference (when no batching and hence no padding is used) that degrade model performance, as demonstrated in Section VI-C.

To mitigate this effect, we introduce the *Masked Layers*, which are a modified version of BN and GAP layers able to account for the zero-padding by applying binary masks during training. These masks ensure that only valid (i.e., non-padded) elements contribute to the batch statistics, shielding BN and GAP from distortions introduced with the HopAugment technique.

a) Masked Batch Normalization: Let $\mathrm{A} \in \mathbb{R}^{N\times C\times U\times V}$ be a tensor representing a batch of intermediate activations, where N is the batch size, C the number of channels, and $U \times V$ the spatial resolution[1]. Standard BN layers compute the per-channel mean μ_c and variance σ_c^2 as:

$$\mu_c = \frac{1}{NUV} \sum_{n,u,v=1}^{N,U,V} a_{nchw}, \tag{1}$$

$$\sigma_c^2 = \frac{1}{NUV} \sum_{n,u,v=1}^{N,U,V} (a_{ncuv} - \mu_c)^2. \tag{2}$$

where a_{ncuv} is the generic element at batch index n, in channel c, height u, and width v.

Due to zero-padding introduced along the time dimension (the tensor width), many entries a_{ncuv} are zero, especially for configurations with larger hop lengths, distorting the computed statistics and leading to mismatched behavior during inference, when no batching and hence no padding is used. To correct such skew, we define a binary mask $\mathrm{M} \in \{0,1\}^{N\times C\times U\times V}$ indicating whether the corresponding input position carries valid data (1) or not (0). The masked BN statistics are then computed as:

$$\mu_c = \frac{1}{\sum_{n,u,v=1}^{N,U,V} m_{ncuv}} \sum_{n,u,v=1}^{N,U,V} m_{ncuv} a_{ncuv}, \tag{3}$$

$$\sigma_c^2 = \frac{1}{\sum_{n,u,v=1}^{N,U,V} m_{ncuv}} \sum_{n,h,w=1}^{N,U,V} m_{ncuv} (a_{ncuv} - \mu_c)^2. \tag{4}$$

This formulation ensures that only valid values contribute to the mean and variance used for normalization, preserving consistent activation statistics across training and inference.

b) Masked Global Average Pooling: In the standard GAP operation, the pooled output z_{nc} for sample n and channel c is given by:

$$z_{nc} = \frac{1}{UV} \sum_{u,v=1}^{U,V} a_{ncuv}. \tag{5}$$

As the average might misestimate the actual feature magnitude due to zero-padding, we introduce the masked version as follows:

$$z_{nc} = \frac{1}{\sum_{u,v=1}^{U,V} m_{ncuv}} \sum_{u,v=1}^{U,V} m_{ncuv} a_{ncuv}. \tag{6}$$

Like for a masked BN layer, the resulting features reflect the true input content, thereby enabling better model learning under variable hop configurations.

To note that masked layers are only needed during training. At inference time, padding is unnecessary since inputs are processed individually, without batching, with a specific hop length. Therefore, the masking operations can be safely omitted from the inference graph, and the deployed model retains the same structure as the original CNN architecture.

[1]Here, U and V are used instead of common W and H to avoid conflict with FE parameters.

V. Experimental Setup

Datasets. We evaluated the performance of the proposed RFC using four widely adopted KWS benchmarks:

- Google Speech Commands datasets versions 1 (GSCv1) and 2 (GSCv2) [51]. GSCv1 collects 64,727 recordings of 30 commands from 1,881 speakers, while GSCv2 contains 105,829 recordings of 35 commands from 2,618 speakers. As done in previous works [11], [12], [14]–[18], we selected ten commands as the target classes ("Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go") and two additional classes for "Silence" (background noise) and "Unknown" (populated with samples from the remaining commands). In both versions, the recordings have a maximum length of 1 s, with a sampling rate of 16 kHz and a bit depth of 16 bits. We followed standard splits for training (80% of all samples), validation (10%), and test (10%).
- Multilingual Spoken Words Corpus (MSWC) [52]. MSWC is a large-scale multilingual dataset designed to support the development and evaluation of KWS for voice-enabled consumer devices. In this work, we used the English subset of the *Microset* version, which comprises approximately 96k one-second utterances spanning 31 target keywords. The recordings originally sampled at 48 kHz with 16-bit resolution were resampled to 16 kHz to match the feature extraction pipeline used for the other datasets. We followed the standard training, validation, and test splits provided by the dataset.
- Hey Snips dataset [53]. This dataset targets wake-word detection for the expression "Hey Snips" and includes a diverse set of English accents, speakers, and recording backgrounds. It consists of approximately 11k positive utterances containing the wake word and 46k negative utterances. All recordings were originally sampled at 16 kHz with 16-bit resolution. We adopted the standard data splits defined in the original dataset.

Front-End and Augmentation. We extracted 40 MFCC features using a frame length of 30 ms, 40 Mel bins ranging from 0 kHz to 8 kHz. During training, we set the proposed HopAugment to generate MFCCs by randomly selecting among three hop length values in the set $\mathcal{H} = \{10\,\mathrm{ms}, 15\,\mathrm{ms}, 20\,\mathrm{ms}\}$. These three values also define the three operating modes of RFC that will be supported at runtime. We also applied augmentation techniques commonly used in KWS [11], [12], [14]–[18], including time shift in the range of -100 ms to 100 ms, random background noise mixing with a probability of 0.8, and SpecAugment [54] with 10 time and 5 frequency masks.

Neural Networks. We primarily focus on the TC-ResNet architecture, which belongs to the family of models based on 1D convolutions along the time dimension. TC-ResNet is widely recognized as a competitive and efficient implementation of time-channel convolutional KWS models and was commonly adopted as a reference architecture by prior work [13], [14], [16]–[19]. Compared with traditional 2D convolutional KWS models, time-channel convolutional architectures reduce the number of parameters and multiply–accumulate operations

while maintaining competitive accuracy. This makes them particularly suitable for deployment on microcontroller-class devices, where computational resources, memory, and energy budgets are tightly constrained. Alternative architectures, such as transformer-based models, typically incur higher computational and memory costs and are usually devised for high-end embedded CPUs. Nevertheless, we also consider two other time-channel 1D convolutional networks in our study, MatchboxNet [13] and TENet [14], which are representative architectures to demonstrate the generality of RFC (Section VI-C5).

Training. The training ran for $80k$ iterations using stochastic gradient descent optimizer with momentum to 0.9, weight decay to $1\mathrm{e}{-3}$, batch size to 100, and a learning rate linearly increasing from zero to 0.1 over the first $2k$ iterations and then decaying to zero with a cosine annealing schedule. Moreover, we applied the proposed masked implementation to the BN and GAP layers of the model.

Quantization. After training, we converted the 32-bit floating-point model to an 8-bit integer representation using post-training quantization (PTQ) for a more efficient deployment. We implemented PTQ using the quantization API provided by the ONNX Runtime[2]. The calibration set used to collect quantization statistics was built randomly, taking 100 samples per class from the training set.

Deployment. The experiments were conducted on an STM32 NUCELO-F767ZI board, featuring a 216-MHz STM32F767ZI ARM Cortex-M7 core, $2\,\mathrm{MB}$ of FLASH, and $512\,\mathrm{kB}$ of SRAM. For the FE processing, we borrowed the MFCC implementation from the CMSIS-DSP library (version 1.16.2) by ARM, which offers signal-processing functions optimized for Cortex-M architectures. The quantized ONNX models were converted into optimized C code using the STEdgeAI-Core[3] toolchain (version 2.20.0). Finally, we cross-compiled the C code using the GNU Arm Embedded Toolchain (`arm-none-eabi-gcc` version 11.3.1) with the optimization flags `-O3` and `-ffast-math` to maximize performance.

Evaluation. For the assessment of the prediction quality, we evaluated the top-1 classification accuracy of the quantized model on the test set. We repeated training, quantization, and evaluation on models initialized with 10 different random seeds and reported the average and standard deviation of accuracy over the 10 runs. This evaluation protocol accounts for the inherent stochastic nature of neural network training and is commonly adopted to obtain statistically reliable performance estimates [12], [13], [16], [18]. To profile latency, we performed on-board measurements for each RFC operating mode by computing the average latency over 100 consecutive inference runs interleaved by a 0.1-second pause following a warm-up phase of 50 runs. Each inference run consists of the FE and NN stages on a one-second input.

[2]https://onnxruntime.ai

[3]https://www.st.com/en/development-tools/stedgeai-core.html

VI. Results

A. RFC vs Topology Scaling

The TC-ResNet-14-15 model trained with our setup supports three runtime operating modes corresponding to hop lengths used during training with HopAugment: $10\,\mathrm{ms}$, $15\,\mathrm{ms}$, and $20\,\mathrm{ms}$. Each mode exploits the same model architecture and weights, allowing the target system to select the operating mode either statically (at compile-time based on hardware constraints) or dynamically (at runtime based on time-varying energy, latency, or power constraints).

Figure 3 shows the accuracy and latency of the three RFC modes on the four benchmark datasets. The three configurations are represented by blue dots: the leftmost point represents the $20\,\mathrm{ms}$ mode (lowest accuracy and latency), while the rightmost corresponds to $10\,\mathrm{ms}$ (highest accuracy and latency). These three configurations define a Pareto frontier, demonstrating that RFC enables flexible trade-offs between quality and efficiency across a wide range of deployment scenarios. The vertical error bars indicate the standard deviation of accuracy over the 10 random seeds.

To evaluate the effectiveness of RFC compared to alternative adaptive approaches, we compare it against Topology Scaling (TS)—a standard strategy that adapts runtime behavior by selecting from a set of independently trained network variants of varying complexity. Specifically, we considered four TC-ResNet variants: TC-ResNet-8, TC-ResNet-8-1.5, TC-ResNet-14, and TC-ResNet-14-1.5 [12]. All models were trained using the same standard pipeline described in Section V, with a fixed hop length of $10\,\mathrm{ms}$ (as in [12]), and without HopAugment and Masked Layers. Each TS variant is trained separately with a dedicated weight set to ensure optimal performance. While some weight-sharing approaches to NN scaling exist (see Section III), they typically result in reduced accuracy. For this reason, despite being less suited for runtime adaptation, TS represents a more challenging baseline for accuracy comparisons.

In Figure 3, the performance of the four TS variants is marked with a red cross. The plots show that RFC achieves a superior latency-accuracy trade-off compared to TS. Across the four datasets, RFC provides up to 31.8% latency reduction at the same accuracy, or 0.30% higher accuracy at equivalent latency, demonstrating a more efficient scaling behavior. These gains stem from RFC's unique ability to jointly modulate the FE and NN workload, unlike TS (and other adaptive inference strategies), which only scales the NN architecture, keeping the FE stage unaltered.

The advantages of compound FE+NN rescaling in RFC can be observed in the latency breakdown shown in Figure 4. The bar chart illustrates the separate contributions of the FE and NN components across the three RFC operating modes, all based on the same TC-ResNet-14-1.5 architecture. As the hop length increases, both FE and NN latencies consistently decrease, yielding up to $1.9\times$ faster processing ($20\,\mathrm{ms}$ vs. $10\,\mathrm{ms}$). This achievement demonstrates that the hop length acts as a more effective knob for coordinated latency scaling across the full KWS pipeline. As shown in Figure 1, with TS, only the NN latency decreases across model variants, while the

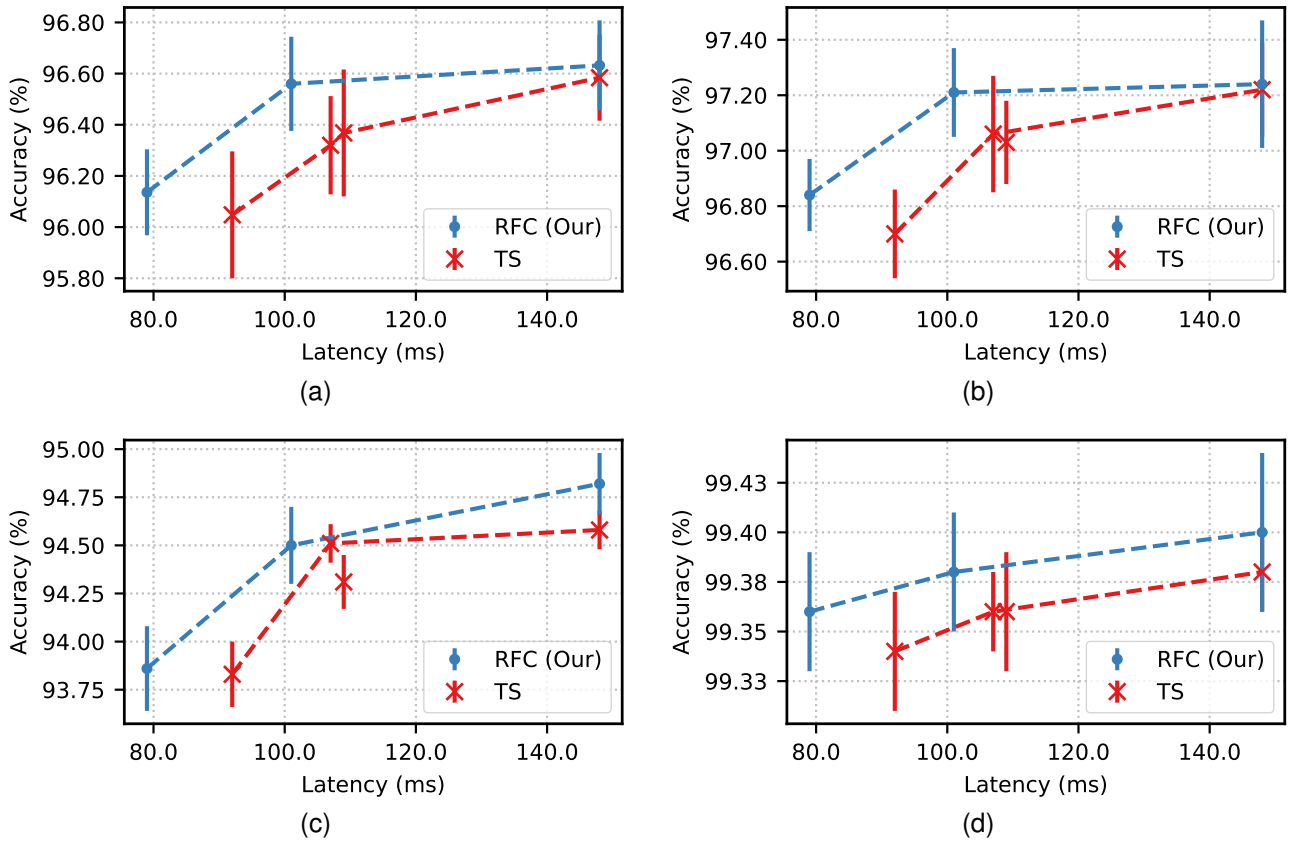


Fig. 3. Accuracy vs. Latency trade-offs obtained with Topology Scaling (TS) and the proposed Runtime Feature Compression (RFC) on an STM32F767ZI MCU. Results on (a) GSCv1, (b) GSCv2, (c) MSWC, (d) Hey Snips.

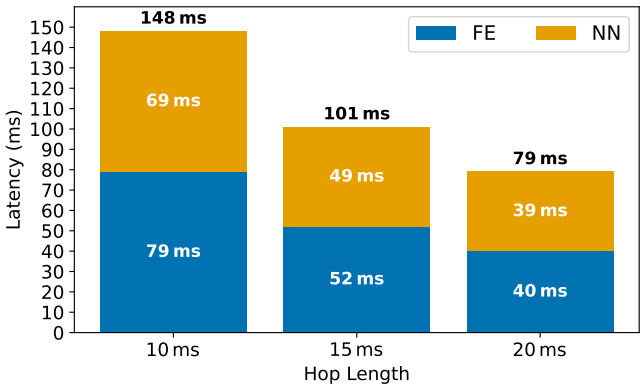


Fig. 4. KWS Latency using TC-ResNet-14-15 with different hop lengths on an STM32F767ZI MCU.

FE latency remains constant. That is because all TS models operate with the same hop length in the feature extraction stage, leading to an unchanged number of STFT frames and the same FE computational cost. The total achievable latency reduction with TS is therefore limited to 1.6× (from TC-ResNet-14-1.5 to TC-ResNet-8) due to the unchanged and still dominant latency contribution of the FE. Moreover, the limited latency scaling in TS comes at the cost of reduced model capacity. Achieving lower latency in TS requires aggressive reduction in the number of network parameters and arithmetic operations, leading to noticeable accuracy drops in the fastest configuration: down to 96.06% on GSCv1 and 96.70% on GSCv2. In contrast, RFC achieves its latency reduction without modifying the network topology across operating modes, thus preserving the full expressive power of the neural model. Instead of sacrificing parameters, RFC leverages redundancy in the input representation by scaling down the number of input frames via increased hop length, significantly reducing the computational demand of both FE and NN stages while preserving accuracy. As a result, RFC enables wider and more effective latency scaling while maintaining higher accuracy: 96.17% on GSCv1 and 96.84% on GSCv2 in its fastest mode.

TABLE II
INPUT FEATURE SHAPE ($F \times T$), NUMBER OF MULTIPLY-AND-ACCUMULATE OPERATIONS (# MACS), AND SRAM REQUIREMENTS WITH DIFFERENT HOP LENGTHS (h_i).

h_i	$F \times T$	# MACs	SRAM
10 ms	40 × 101	7.91 M	13.04 kB
15 ms	40 × 67	5.30 M	10.65 kB
20 ms	40 × 51	4.07 M	9.52 kB

Table II provides a detailed analysis of the impact of RFC on the computational workload of the KWS pipeline, including the input feature shape ($F \times T$), the number of multiply-and-accumulate operations (# MACs), and the peak SRAM requirement during neural inference. Increasing the hop length reduces the number of time frames T from 101 at 10 ms

TABLE III
MEMORY REQUIREMENTS IN TERMS OF FLASH AND SRAM FOR RFC AND T-RECX.

Method	FLASH	SRAM
RFC (Our)	360.03 kB	9.52–13.04 kB
T-RECX	367.38 kB	13.07 kB

to 67 at 15 ms and 51 at 20 ms, resulting in a proportional reduction of the computational cost. The number of MACs decreases from 7.91 M to 5.30 M and 4.07 M, respectively, yielding inference speed-ups that closely follows the hop-length scaling factors ($15\,\text{ms}/10\,\text{ms} \approx 148\,\text{ms}/101\,\text{ms}$ and $20\,\text{ms}/15\,\text{ms} \approx 101\,\text{ms}/79\,\text{ms}$). Due to lower values of T, the intermediate activation tensors are also shrunk accordingly, leading to a reduction in peak SRAM usage from 13.04 kB to 10.65 kB and 9.52 kB. Such memory compression represents an extra savings brought by RFC, which facilitates co-execution with other tasks.

These results demonstrate that the FE, traditionally treated as a static, fixed-cost pipeline stage, can instead be adjusted at runtime, enabling performance and efficiency improvements beyond those achievable through topology scaling alone. The HopAugment and Masked Layers techniques contribute substantially to these remarkable results, as we demonstrate in Section VI-C through dedicated ablation studies.

B. RFC vs. Early-Exit

In this subsection, we compare RFC against T-RECX [28], an adaptive inference strategy based on early exit and designed for resource-constrained edge devices such as MCUs. T-RECX augments the neural network with an intermediate classifier placed along the network depth, enabling early termination of inference when a confidence criterion is satisfied. Specifically, if the confidence score produced by the intermediate classifier exceeds a predefined threshold, its prediction is accepted; otherwise, inference proceeds to the final classifier. The lower the threshold, the larger the probability of early exits, which results in faster, yet less accurate inference.

For a fair comparison, we applied the T-RECX mechanism to the TC-ResNet-14-1.5 architecture. Following the original setup introduced in [28], we placed the early-exit block at 50% of the network depth, and we trained the model using a weighted sum of the cross-entropy losses at the early and final outputs, with the early-exit loss weighted by a factor of 0.3.

Figure 5 reports the achievable accuracy-latency trade-offs for both approaches. The red crosses indicate the operating points of T-RECX, obtained by sweeping the confidence threshold from 0 (leftmost cross) to 1 (rightmost cross) in increments of 0.1. The number of distinct operating points may be smaller due to overlapping configurations, as varying the threshold might not change the early versus late prediction ratio.

The results highlight the main limitation of early-exit strategies such as T-RECX. Moving on the curves from left to right, after a first region characterized by modest speedups and low accuracy loss, the trade-off curve exhibits a sudden drop where

TABLE IV
TOP-1 CLASSIFICATION ACCURACY (IN %) MEASURED ON THE TEST SET (AVERAGE OVER 10 RANDOM SEEDS AND STANDARD DEVIATION) OF TC-RESNET-14-1.5 TRAINED WITH A STANDARD TRAINING (FIRST ROW), ADDING HOPAUGMENT (SECOND ROW), AND ADDING MASKED LAYERS (THIRD ROW) FOR DIFFERENT HOP-LENGTH VALUES. RESULTS ON GSCV1, GSCV2, MSWC, AND HEY SNIPS.

Dataset	Training	Test-time Hop Length 10 ms	15 ms	20 ms
GSCv1	**Standard** h=10 ms	$96.73_{\pm 0.21}$	$93.24_{\pm 0.37}$	$81.93_{\pm 0.58}$
	+ HopAugment	$96.70_{\pm 0.18}$	$95.90_{\pm 0.20}$	$93.62_{\pm 0.40}$
	+ Masked Layers	$96.79_{\pm 0.22}$	$96.70_{\pm 0.23}$	$96.17_{\pm 0.21}$
GSCv2	**Standard** h=10 ms	$97.22_{\pm 0.17}$	$94.77_{\pm 0.24}$	$85.69_{\pm 0.81}$
	+ HopAugment	$97.22_{\pm 0.17}$	$96.70_{\pm 0.22}$	$95.06_{\pm 0.24}$
	+ Masked Layers	$97.24_{\pm 0.23}$	$97.21_{\pm 0.16}$	$96.84_{\pm 0.13}$
MSWC	**Standard** h=10 ms	$94.58_{\pm 0.10}$	$91.96_{\pm 0.23}$	$83.13_{\pm 0.50}$
	+ HopAugment	$94.46_{\pm 0.13}$	$94.33_{\pm 0.15}$	$93.29_{\pm 0.21}$
	+ Masked Layers	$94.82_{\pm 0.16}$	$94.50_{\pm 0.20}$	$93.86_{\pm 0.22}$
Hey Snips	**Standard** h=10 ms	$99.38_{\pm 0.02}$	$98.96_{\pm 0.12}$	$95.45_{\pm 0.87}$
	+ HopAugment	$99.36_{\pm 0.03}$	$99.33_{\pm 0.02}$	$99.28_{\pm 0.03}$
	+ Masked Layers	$99.40_{\pm 0.04}$	$99.38_{\pm 0.03}$	$99.36_{\pm 0.03}$

lowering the confidence threshold leads to marginal latency reductions. This behavior arises because the latency of the front-end stage becomes the dominant contribution, and does not benefit from an early exit of the NN stage. In other words, even in the fastest configuration (corresponding to a confidence threshold of 0), the end-to-end latency is bounded by the front-end cost. To notice that such a fastest T-RECX configuration is still dominated by RFC in both latency and accuracy.

Just like other early-exit strategies, T-RECX inherently lacks predicability. Indeed, the confidence score is input-dependent, and actual savings do change with the input streams, with potential deviations from what is observed during the test phase. In the worst case, when the data are particularly complex, consistently low confidence scores force execution of the full network, resulting in no savings at all. Instead, RFC offers deterministic control, which is paramount for IoT applications requiring predictable latency, energy consumption, and responsiveness.

Table III shows a comparison of the memory footprint in terms of FLASH, used to store network weights, and SRAM, required for intermediate activations. While RFC preserves the original model footprint (360.03 kB of FLASH, 13.04 kB of SRAM for the most accurate configuration), T-RECX introduces memory overhead due to the additional early-exit classifier, incurring 7.35 kB of extra FLASH and up to 3.55 kB of additional SRAM (compared with the fastest the RFC configuration). Even though the overhead may seem negligible, memory is a critical and tightly constrained resource shared with other software components, like buffers, control logic, and peripheral drivers. Therefore, approaches like RFC that enable runtime adaptation without extra memory are better suited for deployments on tiny IoT platforms.

C. Ablation Studies

To further motivate and demonstrate the advantages brought by the proposed training optimizations, we conducted a structured set of targeted ablation studies. The first study focuses on the contributions of HopAugment and Masked Layers,

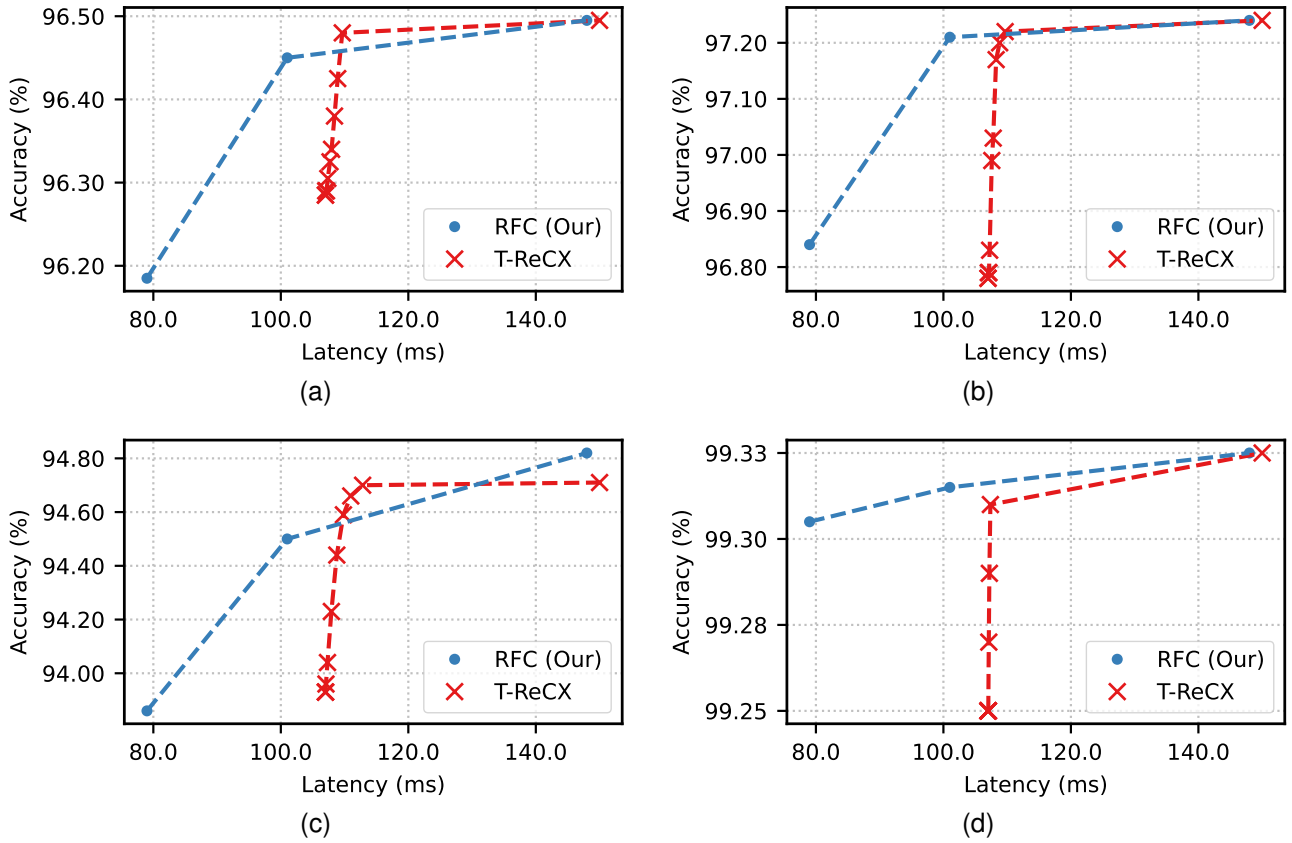


Fig. 5. Accuracy vs. Latency trade-offs obtained with early-exit (T-RECX [28]) and the proposed Runtime Feature Compression (RFC) on an STM32F767ZI MCU. Results on (a) GSCv1, (b) GSCv2, (c) MSWC, (d) Hey Snips.

demonstrating their role in preserving accuracy across different operating modes. The second study compares the proposed HopAugment strategy based on mixed-hop-length batches against an alternative baseline that enforces a uniform hop length within each training batch, thereby assessing the impact of multi-hop length batch composition on the model's ability to generalize across multiple configurations. The third study compares RFC against a baseline where separate weight sets are trained for single hop lengths, highlighting RFC's ability to match with, and eventually improve over models specialized to a single hop configuration. The fourth study evaluates RFC on a larger number of operating modes, assessing its ability to support fine-grained runtime scaling. Finally, the fifth study investigates the applicability of RFC to a different model architecture.

1) HopAugment and Masked Layers: To assess the contribution of HopAugment and Masked Layers, we conducted an ablation study comparing three training configurations:

(i) Standard training: training with fixed hop length of 10 ms, no HopAugment, no Masked Layers;
(ii) Training with HopAugment only;
(iii) Training with both HopAugment and Masked Layers, as detailed in Section IV.

After training, each model was evaluated across all three hop lengths (10 ms, 15 ms, and 20 ms). Table IV reports the average and standard deviation of accuracy across 10 random seeds for each configuration.

The results show the limitations of standard training to supporting hop-length scaling (row **Standard** h=10 ms). For example, on GSCv1, the model trained with fixed hop length achieves 96.73% accuracy at 10 ms but drops sharply to 93.24% and 81.93% at 15 ms and 20 ms, respectively. Such large degradation clearly indicates that conventional training cannot generalize across different hop lengths.

HopAugment improves generalization (row **+ HopAugment**), resulting in less accuracy loss under longer hop lengths. On GSCv1, for example, accuracy improves to 95.90% at 15 ms and 93.62% at 20 ms, yet a notable performance gap remains. This is due to a mismatch in activation statistics in the output of BN and GAP layers, caused by the zero padding used in training that is absent at inference time.

The introduction of the Masked Layers effectively fixes these issues (row **+ Masked Layers**). The resulting model achieves consistent accuracy across all operating modes: from 96.79% to 96.17% on GSCv1, and from 97.24% to 96.84% on GSCv2, showing minimal degradation with hop-length scaling.

The results confirm that HopAugment alone is insufficient to maintain competitive accuracy across all hop-length configurations. Masked Layers are important to preserve compatible normalization and pooling statistics under dynamically compressed features. The combined use of HopAugment and Masked Layers is key to enabling RFC to generalize properly. These findings also reveal that HopAugment fundamentally differs from conventional augmentation techniques, like resiz-

TABLE V
TOP-1 CLASSIFICATION ACCURACY (IN %) MEASURED ON THE TEST SET (AVERAGE OVER 10 RANDOM SEEDS AND STANDARD DEVIATION) OF TC-RESNET-14-1.5 TRAINED WITH MIXED-HOP-LENGTH BATCHES AND UNIFORM-HOP-LENGTH BATCHES (SECOND ROW) FOR DIFFERENT HOP-LENGTH VALUES. RESULTS ON GSCV1 AND GSCV2.

Dataset	Training Batches	Test-time Hop Length		
		10 ms	15 ms	20 ms
GSCv1	Mixed (RFC)	$96.79_{\pm 0.22}$	$96.70_{\pm 0.23}$	$96.17_{\pm 0.21}$
	Uniform	$96.57_{\pm 0.19}$	$95.91_{\pm 0.24}$	$95.44_{\pm 0.13}$
GSCv2	Mixed (RFC)	$97.24_{\pm 0.23}$	$97.21_{\pm 0.16}$	$96.84_{\pm 0.13}$
	Uniform	$97.26_{\pm 0.14}$	$96.54_{\pm 0.19}$	$96.45_{\pm 0.21}$

ing, which typically produce features with similar statistics and do not require such kind of mechanisms.

2) Training with Mixed- vs. Uniform-Hop-Length Batches: A potential alternative to the proposed training strategy consists of avoiding mixed-size features by enforcing a uniform hop length per batch. In this setting, all samples within a training batch share the same hop length, avoiding the need for feature padding and Masked Layers. Batches cycle over the target hop lengths. For instance, batch 0, 10 ms, batch 1, 15 ms, batch 2, 20 ms, (and then loop) batch 3, 10 ms. We implemented this Uniform-Hop-Length strategy, whose results are reported in Table V together with those obtained with our Mixed-Hop-Length strategy. For both methods, the collected numbers refer to the average and standard deviation of accuracy over 10 random seeds.

Uniform-Hop-Length batching achieves comparable, and in some cases slightly higher, accuracy working under the smallest hop length (e.g., -0.22% on GSCv1 and $+0.02\%$ on GSCv2 at 10 ms). However, it exhibits substantial degradation when tested with larger hop lengths, namely, for faster operating modes. Specifically, accuracy drops by 0.79% and 0.73% on GSCv1 at 15 ms and 20 ms, respectively, and by 0.67% and 0.39% on GSCv2.

These findings suggest that training with mixed hop lengths within each batch is important to learning model weights that generalize robustly across the available operating modes. While uniform hop length simplifies tensor handling, it prevents robust learning from variable configurations, resulting in weaker generalization at larger hop lengths. This further motivates the joint use of HopAugment and Masked Layers to enable effective runtime hop-length scaling.

3) RFC vs. Specialized Models: RFC operates using a single, shared weight set across all supported hop lengths. To evaluate its effectiveness, we compared RFC against a set of models individually trained for a fixed hop length. Specifically, we trained three separate instances of TC-ResNet-14-1.5 using hop lengths of 10 ms, 15 ms, and 20 ms, and the standard training procedure, namely, without HopAugment and Masked Layers. Table VI summarizes the results. Each model is evaluated under the three hop-length settings at inference time. For ease of comparison, we also included the RFC results (already reported in Figs. 3a and 3b). Two key insights emerge from the analysis of the collected results.

First, all models trained with a fixed hop length (first three rows in each section) exhibit significant accuracy degradation when evaluated with a different hop length than used in training. For example, on GSCv1 (top section of the table), the model trained with 15 ms-hop length achieves its highest accuracy when evaluated with 15 ms (96.56%), but performance drops to 93.15% at 10 ms and 94.77% with 20 ms. The drop is quite evident for 10 ms, where a shorter hop length, despite delivering finer features, results in worse performance due to a mismatch in temporal granularity between training and test. Similar trends are observed on GSCv2, MSWC and Hey Snips. These findings reveal that models trained with a fixed hop length lack robustness to hop-length variations and are therefore unsuited for runtime adaptation.

TABLE VI
TOP-1 CLASSIFICATION ACCURACY (%) COMPARISON BETWEEN RFC AND INDIVIDUALLY RETRAINED MODELS SPECIALIZED TO FIXED HOP LENGTHS (10 ms, 15 ms, AND 20 ms). RESULTS ON GSCV1, GSCV2, MWSC, AND HEY SNIPS.

Dataset	Training	Test-time Hop Length		
		10 ms	15 ms	20 ms
GSCv1	Standard h=10 ms	$96.73_{\pm 0.21}$	$93.24_{\pm 0.37}$	$81.93_{\pm 0.58}$
	Standard h=15 ms	$93.15_{\pm 0.49}$	$96.56_{\pm 0.20}$	$94.77_{\pm 0.27}$
	Standard h=20 ms	$81.56_{\pm 1.07}$	$94.95_{\pm 0.26}$	$96.29_{\pm 0.27}$
	RFC	$96.79_{\pm 0.22}$	$96.70_{\pm 0.23}$	$96.17_{\pm 0.21}$
GSCv2	Standard h=10 ms	$97.22_{\pm 0.17}$	$94.77_{\pm 0.24}$	$85.69_{\pm 0.81}$
	Standard h=15 ms	$94.15_{\pm 0.22}$	$97.18_{\pm 0.12}$	$96.04_{\pm 0.22}$
	Standard h=20 ms	$81.83_{\pm 1.16}$	$95.46_{\pm 0.23}$	$96.93_{\pm 0.15}$
	RFC	$97.24_{\pm 0.23}$	$97.21_{\pm 0.16}$	$96.84_{\pm 0.13}$
MSWC	Standard h=10 ms	$94.58_{\pm 0.10}$	$91.96_{\pm 0.23}$	$83.13_{\pm 0.50}$
	Standard h=15 ms	$90.44_{\pm 0.24}$	$94.52_{\pm 0.13}$	$93.17_{\pm 0.09}$
	Standard h=20 ms	$81.36_{\pm 0.77}$	$92.46_{\pm 0.15}$	$94.13_{\pm 0.18}$
	RFC	$94.82_{\pm 0.16}$	$94.50_{\pm 0.20}$	$93.86_{\pm 0.22}$
Hey Snips	Standard h=10 ms	$99.38_{\pm 0.02}$	$98.96_{\pm 0.12}$	$95.45_{\pm 0.87}$
	Standard h=15 ms	$99.23_{\pm 0.04}$	$99.38_{\pm 0.03}$	$99.30_{\pm 0.02}$
	Standard h=20 ms	$98.17_{\pm 0.15}$	$99.25_{\pm 0.06}$	$99.37_{\pm 0.04}$
	RFC	$99.40_{\pm 0.04}$	$99.38_{\pm 0.03}$	$99.36_{\pm 0.03}$

Second, and perhaps more remarkably, the RFC-trained model not just holds high accuracy across all hop lengths, but often outperforms models specialized for each configuration, where the train-time and test-time hop lengths do match. For instance, on GSCv1 with 10 ms, RFC achieves 96.79% accuracy, exceeding the 96.73% achieved by the fixed-hop model. With 15 ms, RFC achieves 96.70% vs. 96.56% for the fixed-hop model. The only exception is with 20 ms, where the fixed-hop model slightly outperforms RFC (96.29% vs. 96.17%). The same pattern is observed on GSCv2.

The above results demonstrate that, compared to static, hop length-specific training, the proposed RFC-enhanced training enables effective runtime adaptivity, without compromising, and sometimes improving, accuracy. This highlights the strength of HopAugment and Masked Layers, which enable single models to generalize across varying hop lengths without performance penalties.

4) Number of Operating Modes: This subsection aims to assess whether the proposed approach gets stable under a finer trade-off with more operating modes. We evaluated RFC using five operating modes, corresponding to hop lengths $h \in \{10, 12.5, 15, 17.5, 20\}$ ms. Figure 6 shows the resulting accuracy-latency trade-offs on GSCv1 and GSCv2, using topology scaling (TS) as the baseline. RFC achieves a superior trade-off even with a larger number of operating modes,

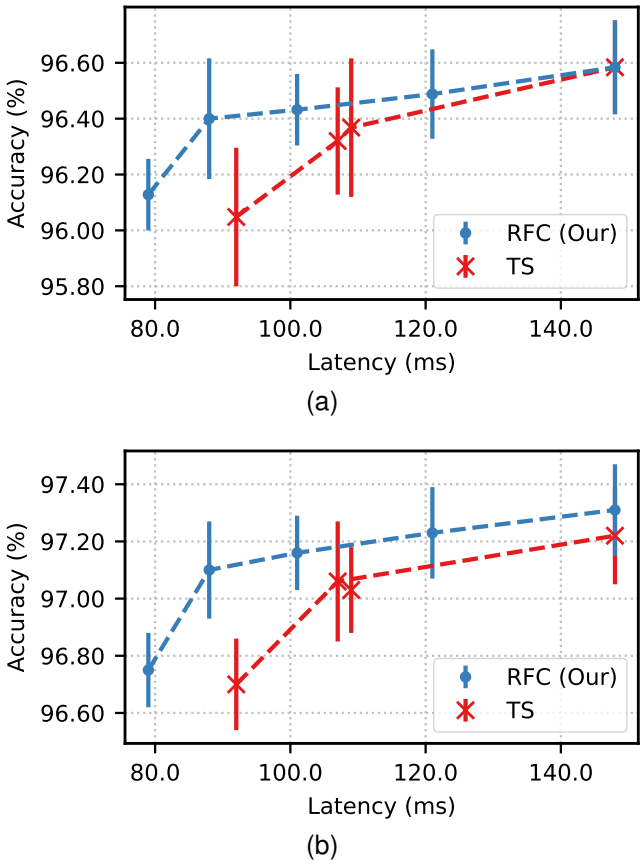


Fig. 6. Accuracy vs. Latency trade-offs obtained with Topology Scaling (TS) and RFC with five operating modes on an STM32F767ZI MCU. Results on (a) GSCv1 and (b) GSCv2.

demonstrating effective fine-grained runtime scaling.

5) RFC with Other Model Architectures: While the previous studies were conducted on the TC-ResNet architecture, well known for its favorable balance between accuracy and efficiency, and therefore commonly adopted for deployment on resource-constrained devices, RFC is conceived as an orthogonal strategy that can be seamlessly applied to other model architectures. To demonstrate its general applicability, we conducted additional experiments using MatchboxNet [13], a more recent KWS model based on time-channel separable convolutions, and TENet [14], a time-channel convolutional architecture built with inverted bottleneck blocks. The MatchboxNet family includes three variants with increasing depth and width (MatchboxNet3x1x64, MatchboxNet3x2x64, and MatchboxNet6x2x64), while the TENet family comprises four variants (TENet6Narrow, TENet12Narrow, TENet6, TENet12), which serve as the backbones for the topology scaling (TS) strategy. Figures 7 and 8 report the corresponding accuracy–latency trade-offs, comparing TS variants against the RFC-enhanced MatchboxNet6x2x64 and TENet12 models, respectively. The results confirm that RFC delivers superior quality compared to TS, preserving higher accuracy while enabling faster inference. These findings demonstrate that the benefits of RFC extend across different neural architectures, proving its broad applicability.

VII. Conclusion

This work introduced Runtime Feature Compression (RFC), a novel strategy that enhances the adaptability of keyword spotting (KWS) systems deployed in energy-constrained and latency-sensitive IoT environments. Unlike conventional adaptive approaches that rescale only the neural network architecture, RFC enables joint runtime rescaling of both the front-end and the neural network by dynamically adjusting the hop length in the early stages of the processing pipeline. This compound scaling mechanism allows a single trained model to operate efficiently across multiple accuracy-latency operating points without requiring multiple weight sets or architectural modifications.

Extensive experimental evaluations on four widely used KWS benchmarks demonstrated that RFC consistently achieves superior latency-accuracy trade-offs compared to state-of-the-art adaptive methods. In particular, RFC attains up to 31.8% latency reduction without accuracy degradation, or up to 0.30% higher accuracy under equivalent latency constraints. These gains stem from RFC's ability to reduce the computational load of both feature extraction and neural inference in a coordinated manner, rather than limiting adaptation to the classifier alone.

A comprehensive set of ablation studies further validated the robustness and generality of the proposed approach. The ablations showed that HopAugment and Masked Layers are both essential to preserve accuracy under variable hop-length configurations, that mixed-hop-length batching is critical for effective generalization across operating modes, and that RFC can match or outperform models specialized to fixed hop lengths while using a single shared weight set. Additional studies demonstrated that RFC supports fine-grained runtime scaling with an increased number of operating modes and generalizes effectively to different neural network architectures.

References

[1] S. Ahmed, I. Shumailov, N. Papernot, and K. Fawaz, "Towards more robust keyword spotting for voice assistants," in *Proc. 31st USENIX Secur. Symp. (USENIX Security 22)*, 2022, pp. 2655–2672.

[2] Z. Cai, Y. Ma, and F. Lu, "Robust dual-modal speech keyword spotting for XR headsets," *IEEE Trans. Vis. Comput. Graph.*, vol. 30, no. 5, pp. 2507–2516, May 2024.

[3] I. López-Espejo, Z. Tan, and J. Jensen, "Improved external speaker-robust keyword spotting for hearing assistive devices," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 28, pp. 1233–1247, 2020.

[4] M. Rusci, F. Paci, M. Fariselli, E. Flamand, and T. Tuytelaars, "Self-learning for personalized keyword spotting on ultralow-power audio sensors," *IEEE Internet Things J.*, vol. 12, no. 8, pp. 10 210–10 221, Apr. 2025.

[5] B. Mao, F. Tang, Y. Kawamoto, and N. Kato, "AI models for green communications towards 6g," *IEEE Commun. Surv. Tutorials*, vol. 24, no. 1, pp. 210–247, 1st Quart. 2022.

[6] Y. Sun, J. Liu, J. Wang, Y. Cao, and N. Kato, "When machine learning meets privacy in 6g: A survey," *IEEE Commun. Surv. Tutorials*, vol. 22, no. 4, pp. 2694–2724, 4th Quart. 2020.

[7] B. Mao, J. Liu, Y. Wu, and N. Kato, "Security and privacy on 6g network edge: A survey," *IEEE Commun. Surv. Tutorials*, vol. 25, no. 2, pp. 1095–1127, 2nd Quart. 2023.

[8] T. Li, Y. Ma, and T. Endoh, "From algorithm to module: Adaptive and energy-efficient quantization method for edge artificial intelligence in iot society," *IEEE Trans. Ind. Informatics*, vol. 19, no. 8, pp. 8953–8964, 2023.

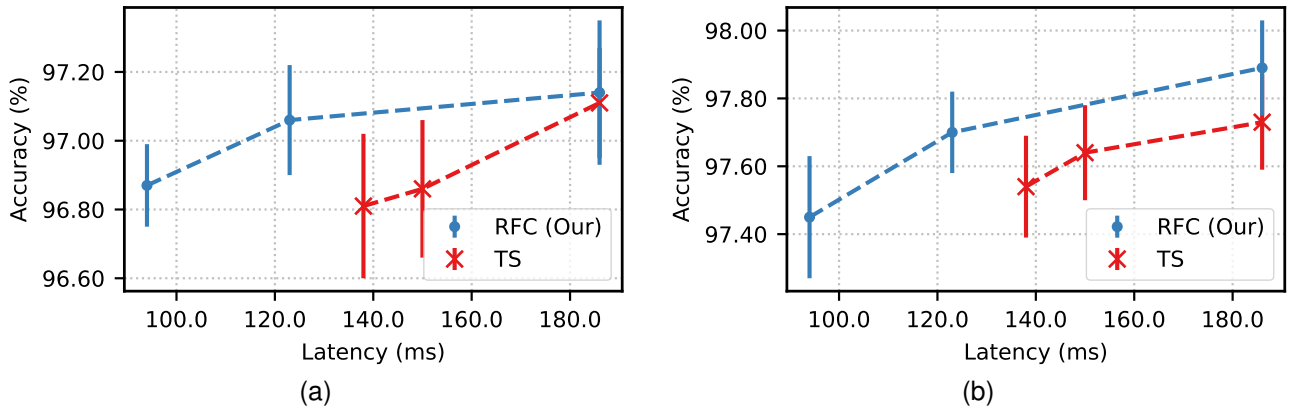


Fig. 7. Accuracy vs. Latency with Topology Scaling (TS) and RFC on an STM32F767ZI MCU. Results for MatchboxNet on (a) GSCv1 and (b) GSCv2.

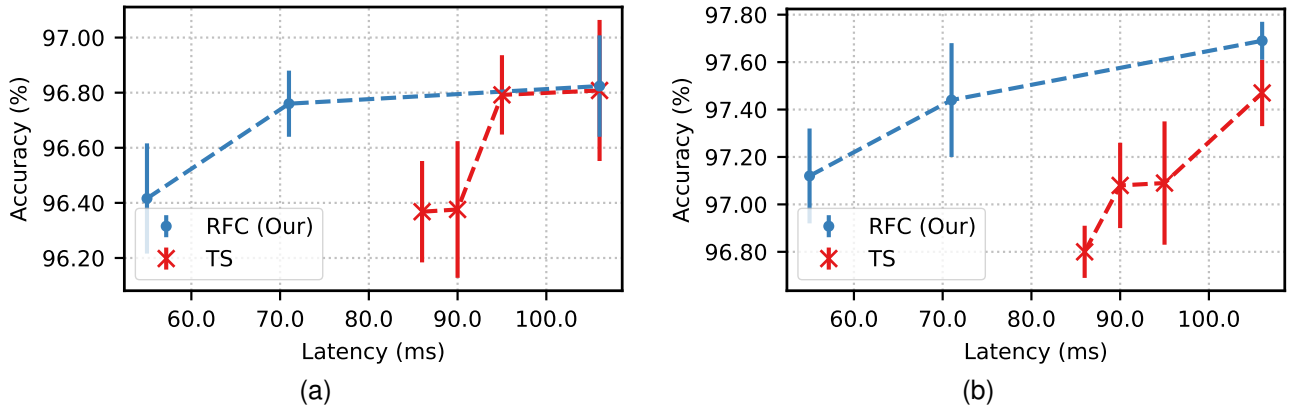


Fig. 8. Accuracy vs. Latency with Topology Scaling (TS) and RFC on an STM32F767ZI MCU. Results for TENet on (a) GSCv1 and (b) GSCv2.

[9] M. Kato, T. K. Rodrigues, and N. Kato, "Breakout local search solution to the offloading decision problem in a multi-access edge computing cloud-enabled network," *IEEE Trans. Emerg. Top. Comput.*, vol. 13, no. 3, pp. 1328–1338, 2025.

[10] T. Li, Y. Ma, K. Yoshikawa, O. Nomura, and T. Endoh, "Energy-efficient convolution module with flexible bit-adjustment method and ADC multiplier architecture for industrial iot," *IEEE Trans. Ind. Informatics*, vol. 18, no. 5, pp. 3055–3065, 2022.

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

[12] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," in *Proc. Interspeech*, 2019, pp. 3372–3376.

[13] S. Majumdar and B. Ginsburg, "Matchboxnet: 1d time-channel separable convolutional neural network architecture for speech commands recognition," in *Proc. Interspeech*, 2020, pp. 3356–3360.

[14] X. Li, X. Wei, and X. Qin, "Small-footprint keyword spotting with multi-scale temporal convolution," in *Proc. Interspeech*, 2020, pp. 1987–1991.

[15] C. R. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," in *Proc. Mach. Learn. Syst. (MLSys)*, 2021, pp. 1–16.

[16] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," in *Proc. Interspeech*, 2021, pp. 4538–4542.

[17] E. Park, D. Ahn, and H. Kim, "Reptor: Re-parameterizable temporal convolution for keyword spotting via differentiable kernel search," in *Proc. Interspeech*, 2024, pp. 4518–4522.

[18] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," in *Proc. Interspeech*, 2021, pp. 4249–4253.

[19] K. Ding, M. Zong, J. Li, and B. Li, "LETR: A lightweight and efficient transformer for keyword spotting," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process (ICASSP)*, 2022, pp. 7987–7991.

[20] I. López-Espejo, Z. Tan, J. H. L. Hansen, and J. Jensen, "Deep spoken keyword spotting: An overview," *IEEE Access*, vol. 10, pp. 4169–4199, 2022.

[21] J. S. P. Giraldo and M. Verhelst, "Hardware acceleration for embedded keyword spotting: Tutorial and survey," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 6, pp. 1–25, Nov. 2021.

[22] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE 23rd Int. Conf. Pattern Recognit. (ICPR)*, 2016, pp. 2464–2469.

[23] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proc. Eur. Conf. Comp. Vis (ECCV)*, 2018, pp. 3–18.

[24] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proc. Eur. Conf. Comp. Vis (ECCV)*, 2018, pp. 409–424.

[25] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu, "Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[26] A. Cipolletta, V. Peluso, A. Calimera, M. Poggi, F. Tosi, F. Aleotti, and S. Mattoccia, "Energy-quality scalable monocular depth estimation on low-power cpus," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 25–36, Jan. 2022.

[27] H. Li, T. Thormundsson, I. Poupyrev, and N. Gillian, "Gated compression layers for efficient always-on models," *arXiv:2303.08970*, 2023.

[28] N. P. Ghanathe and S. Wilton, "T-recx: Tiny-resource efficient convolutional neural networks with early-exit," in *Proc. 20th ACM Int. Conf. Comput. Frontiers (CF)*, 2023, pp. 123–133.

[29] J. Jo, G. Kim, S. Kim, and J. Park, "Locoexnet: Low-cost early exit network for energy efficient CNN accelerator design," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 12, pp. 4909–4921, Dec. 2023.

[30] M. Sponner, L. Servadei, B. Waschneck, R. Wille, and A. Kumar, "Leveraging temporal patterns: Automated augmentation to create temporal early exit networks for efficient edge AI," *IEEE Access*, vol. 12, pp. 169 787–169 804, 2024.
[31] H. Tann, S. Hashemi, R. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proc. 11th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, 2016, pp. 1–10.
[32] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. 30th Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 2181–2191.
[33] X. Gao, Y. Zhao, Łukasz Dudziak, R. Mullins, and C. zhong Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–14.
[34] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.
[35] J. Ma and S. Reda, "Wenet: Configurable neural network with dynamic weight-enabling for efficient inference," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2023, pp. 1–6.
[36] D. J. Pagliari, E. Macii, and M. Poncino, "Dynamic bit-width reconfiguration for energy-efficient deep learning hardware," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2018, p. 47.
[37] V. Peluso and A. Calimera, "Scalable-effort convnets for multilevel classification," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.
[38] ——, "Energy-driven precision scaling for fixed-point convnets," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, 2018, pp. 113–118.
[39] H. Kong, D. Liu, S. Huai, X. Luo, R. Subramaniam, C. Makaya, Q. Lin, and W. Liu, "Edgecompress: Coupling multidimensional model compression and dynamic inference for edgeai," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 12, pp. 4657–4670, Dec. 2023.
[40] M. Shahnawaz, E. Plebani, I. Guaneri, D. Pau, and M. Marcon, "Studying the effects of feature extraction settings on the accuracy and memory requirements of neural networks for keyword spotting," in *Proc. IEEE 8th Int. Conf. Consum. Electron.-Berlin (ICCE-Berlin)*, 2018, pp. 1–6.
[41] W. Toussaint, A. Mathur, A. Y. Ding, and F. Kawsar, "Characterising the role of pre-processing parameters in audio-based embedded machine learning," in *Proc. 19th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2021, pp. 439–445.
[42] A. Soyyigit, S. Yao, and H. Yun, "VALO: A versatile anytime framework for lidar-based object detection deep neural networks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 43, no. 11, pp. 4045–4056, Nov. 2024.
[43] P. Busia, G. Deriu, L. Rinelli, C. Chesta, L. Raffo, and P. Meloni, "Target-aware neural architecture search and deployment for keyword spotting," *IEEE Access*, vol. 10, pp. 40 687–40 700, 2022.
[44] G. Zema, V. Peluso, A. Calimera, and E. Macii, "Concurrent pipeline stages optimization for embedded keyword spotting," in *Proc. IEEE World AI IoT Congr. (AIIoT)*, 2023, pp. 427–433.
[45] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 1, pp. 1–28, Jan. 2020.
[46] W. Lou, L. Xun, A. Sabet, J. Bi, J. S. Hare, and G. V. Merrett, "Dynamic-ofa: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 3110–3118.
[47] Z. Taufique, A. Vyas, A. Miele, P. Liljeberg, and A. Kanduri, "Tango: Low latency multi-dnn inference on heterogeneous edge platforms," in *Proc. IEEE Int. Conf. Comput. Des. (ICCD)*, 2024, pp. 300–307.
[48] R. G. Rizzo, V. Peluso, and A. Calimera, "TVFS: topology voltage frequency scaling for reliable embedded convnets," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 68, no. 2, pp. 672–676, Feb. 2021.
[49] M. Sponner, B. Waschneck, and A. Kumar, "Adapting neural networks at runtime: Current trends in at-runtime optimizations for deep learning," *ACM Comput. Surv.*, vol. 56, no. 10, p. 248, Oct. 2024.
[50] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2016, pp. 475–480.
[51] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv:1804.03209*, 2018.
[52] M. Mazumder, S. Chitlangia, C. R. Banbury, Y. Kang, J. Ciro, K. Achorn, D. Galvez, M. Sabini, P. Mattson, D. Kanter, G. Diamos, P. Warden, J. Meyer, and V. J. Reddi, "Multilingual spoken words corpus," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track (Round 2)*, 2021, pp. 1–17.
[53] A. Coucke, M. Chlieh, T. Gisselbrecht, D. Leroy, M. Poumeyrol, and T. Lavril, "Efficient keyword spotting using dilated convolutions and gating," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process (ICASSP)*, 2019, pp. 6351–6355.
[54] D. S. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," in *Proc. Interspeech*, 2019, pp. 2613–2617.



Valentino Peluso (Member, IEEE) received the M.Sc. degree in electronic engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino, Turin, Italy, in 2015 and 2020, respectively. He is currently an Assistant Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include design automation of digital circuits, low-power embedded systems, energy-efficient edge artificial intelligence, and distributed and federated learning.



Andrea Calimera (Member, IEEE) received the M.Sc. degree in electronic engineering and a Ph.D. in computer engineering from the Politecnico di Torino, Turin, Italy, in 2007 and 2011, respectively. He is a Full Professor of computer engineering at the Politecnico di Torino. His research interests cover the design automation of digital circuits and embedded systems, focusing on synthesis and optimization techniques for low-power, energy-efficient, and reliable applications.



Enrico Macii (Fellow, IEEE) received the Laurea degree in electrical engineering from the Politecnico di Torino, Turin, Italy, the Laurea degree in computer science from the Università di Torino, Turin, and the Ph.D. degree in computer engineering from the Politecnico di Torino received respectively in 1990, 1991, and 1995. He is currently a Full Professor of Computer Engineering with the Politecnico di Torino. His research interests include the design of electronic digital circuits and systems, with a particular emphasis on low-power consumption aspects.



Paolo Montuschi (Fellow, IEEE) is a full professor with the Department of Control and Computer Engineering, Rector's Delegate for Information Systems, and a past member of the Board of Governors at Politecnico di Torino, Italy. His research interests include computer arithmetic, computer architectures, and intelligent systems. He is an IEEE Fellow, a life member of the International Academy of Sciences in Turin, and of HKN, the Honor Society of IEEE. He serves as the Editor-in-Chief of the IEEE Transactions on Emerging Topics in Computing, the 2020-23 Chair of the IEEE TAB/ARC, and a member of the IEEE Awards Board. Previously, he served in a number of positions, including the Editor-in-Chief of the IEEE Transactions on Computers (2015-18), the 2017-20 IEEE Computer Society Awards Committee Chair, a Member-at-Large of IEEE PSPB (2015-20), and as the Chair of its Strategic Planning Committee (2019-20).