UR2M: Uncertainty and Resource-Aware Event Detection on Microcontrollers

Hong Jia*, Young D. Kwon*, Dong Ma†, Nhat Pham‡, Lorena Qendro§, Tam Vu¶ and Cecilia Mascolo*
*University of Cambridge, Cambridge, UK †Singapore Management University, Singapore
‡Cardiff University, Cardiff, UK §Nokia Bell Labs, Cambridge, UK ¶University of Colorado Boulder, Colorado, US
{hj359, ydk21}@cam.ac.uk, dongma@smu.edu.sg, phamn@cardiff.ac.uk,
lorena.qendro@nokia-bell-labs.com, tam.vu@colorado.edu, cm542@cam.ac.uk

***Abstract*—Traditional machine learning techniques are prone to generating inaccurate predictions when confronted with shifts in the distribution of data between the training and testing phases. This vulnerability can lead to severe consequences, especially in applications such as mobile healthcare. Uncertainty estimation has the potential to mitigate this issue by assessing the reliability of a model's output. However, existing uncertainty estimation techniques often require substantial computational resources and memory, making them impractical for implementation on microcontrollers (MCUs). This limitation hinders the feasibility of many important on-device wearable event detection (WED) applications, such as heart attack detection.**

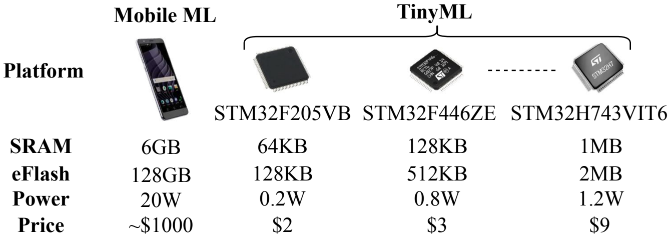
In this paper, we present UR2M, a novel Uncertainty and Resource-aware event detection framework for MCUs. Specifically, we (i) develop an uncertainty-aware WED based on evidential theory for accurate event detection and reliable uncertainty estimation; (ii) introduce a cascade ML framework to achieve efficient model inference via early exits, by sharing shallower model layers among different event models; (iii) optimize the deployment of the model and MCU library for system efficiency. We conducted extensive experiments and compared UR2M to traditional uncertainty baselines using three wearable datasets. Our results demonstrate that UR2M achieves up to 864% faster inference speed, 857% energy-saving for uncertainty estimation, 55% memory saving on two popular MCUs, and a 22% improvement in uncertainty quantification performance. UR2M can be deployed on a wide range of MCUs, significantly expanding real-time and reliable WED applications.

***Index Terms*—Uncertainty, Event Detection, Efficiency, Microcontrollers**

I. Introduction

With advancements in pervasive, low-power, and embedded sensors, a range of human physiological signals can be collected and continuously analyzed. Empowered by machine learning (ML), especially deep learning (DL), these sensors provide great opportunities for a plethora of wearable event detection (WED) applications, such as the detection of stress levels [1], blood pressure [2], or respiratory illnesses [3]. Recently, deploying ML models directly on microcontrollers (MCUs) has attracted tremendous attention due to their potential to improve user privacy and computational latency in WED, especially under unstable network conditions [4]. However, as shown in Figure 1, designing and deploying efficient WED models on MCUs is challenging due to their limited memory space and battery life, especially in comparison to mobile phones [4].

Furthermore, many existing WED models prioritize enhancing classification accuracy while overlooking the importance of prediction reliability [5], which is crucial in fields like health. Reliability is quantified as *uncertainty*, indicating the trustworthiness of the classification results [6]. Factors such as hardware differences, environmental variations, data collection methods, and sensor degradation can lead to distribution shifts between training and testing data (data uncertainty) or unseen data (model uncertainty [7]), reducing the reliability of WED models.



	Mobile ML	TinyML		
Platform		STM32F205VB	STM32F446ZE	STM32H743VIT6
SRAM	6GB	64KB	128KB	1MB
eFlash	128GB	128KB	512KB	2MB
Power	20W	0.2W	0.8W	1.2W
Price	~$1000	$2	$3	$9

Fig. 1: Memory and power comparison between a typical mobile phone and microcontrollers.

Several methods for quantifying uncertainty have been investigated. Bayesian Neural Networks (BNNs), a prominent approach for uncertainty estimation, quantify uncertainty by estimating posteriors over model weights [8]. However, BNNs entail substantial computational expenses [9]. Although approximation techniques such as Monte Carlo dropout (MCDP) [10] and deep ensembles [7] have been proposed, these methods still require ensembling multiple models and various inference steps, which introduce intensive computational and memory demands, as well as increased latency. Recent research has also introduced deterministic models that require only one forward pass, making them more efficient but at the cost of lower accuracy [11]. As a result, integrating reliable uncertainty could pose additional complexities in the design and deployment of trustworthy WED models on MCUs.

Lastly, existing works demonstrate inefficiency in supporting multi-event detection on MCUs, as they typically employ individual models for each event to ensure reusability across different applications or use cases and to optimize efficiency for each model [12]. However, wearable devices often require the simultaneous detection of multiple events. For instance, a single electroencephalography (EEG) input might be utilized to concurrently detect the brain's alpha wave (event 1) for a guided-meditation application, and beta wave (event 2) for a focus monitoring application. Additionally, executing multiple inferences (encompassing both prediction and uncertainty estimation) for varied events can be resource-intensive, potentially rendering WED deployment on MCUs impracticable due to memory constraints.

To address the aforementioned challenges, we propose an efficient uncertainty estimation approach based on evidential deep learning (EDL) and cascade learning. Specifically, (i) EDL is designed to predict a distribution, parameterized by a vector, instead of providing a point prediction through a single DL model, which allows for the direct prediction of event detection and its associated uncertainty via a single inference. (ii) For each event (intra-event), we consider three models of varied depths (i.e., shallow, medium, and deep); herein, deeper models are stacked upon shallower ones, meaning the lower layers are shared. A classifier layer (termed a "head") is appended to each model. This design adheres to the observation that some testing samples, particularly those near the center of the training sample distribution, do not require a full pass through the deep model to ensure a reliable prediction [13]. Consequently, early exits can be employed to enhance computational cost-effectiveness and inference speed, with uncertainty chosen as the criterion for an early exit to ensure the reliability of the prediction. (iii) For multiple events (inter-event) using the same input, we propose the sharing of all layers for feature extraction and the training of individual classification layers (referred to as "multi-heads"). As a result, our framework can be effortlessly scaled to multiple events with minimal memory overhead, since only the heads need to be added. Additionally, reusing shared layers for different events reduces computation time and cost.

We further apply three techniques to improve the efficiency of our approach during implementation. First, we implement an architecture search to find the optimal model structure automatically (e.g., number of model layers and size of channels) for specific WED tasks based on recent success models designed for MCUs [14]. Second, we conduct scalar quantization of the model weights into 8-bit integers to decrease the model size and further save memory. Third, to reduce the memory consumption of the deep learning library, we remove unnecessary components that are not utilized in our models. Finally, we conduct comprehensive experiments with two MCU platforms to demonstrate the effectiveness of the proposed approach.

To summarize, we make the following contributions:

- We propose a cascade model architecture with intra-event and inter-event layer sharing to enable efficient multi-event detection. We also conduct efficient architecture search, model compression, and library optimization to improve system efficiency (§V-§VI).
- We propose a novel uncertainty-aware learning paradigm based on evidential theory for efficient and reliable WED uncertainty estimation on MCUs (§IV).
- We conduct extensive experiments on three popular wearable datasets and implement our framework on two off-the-shelf MCUs, including STM32F446ZE and STM32H747F7, with limited SRAM memory (128KB and 512KB, respectively). Our evaluation shows that the proposed framework performs up to 864% better inference speed and 857% energy saving compared to uncertainty baselines. The approach also saves 55% of memory compared with existing uncertainty estimation baselines (§VII-§VIII), enabling the deployment of WED models on MCUs with limited memory (e.g., STM32F205VB with 64KB SRAM).

II. Related Works

This section briefly discusses the literature on machine learning on MCUs, event detection on resource-constrained devices, and efficient methods for uncertainty estimation.

Tiny machine learning on MCUs. Tiny Machine Learning [14] (TinyML) aims to execute deep learning models locally on extremely resource-constrained devices such as MCUs. Recent studies have concentrated on optimizing network architectures considering constraints such as limited memory, energy, FLOPs [4], and processor speed [15]. However, these approaches focus solely on classification accuracy, treating them as single-point predictions without considering uncertainty estimation. In contrast, we further include uncertainty estimation of the desired predictions to enable a more reliable WED.

Event detection on resource-constrained devices. Recent years have seen a surge in research focused on event detection using wearables, exploring various sensing modalities including image [16], audio [17], electrocardiogram (ECG) [18], and others. However, most existing WED approaches only utilize wearables for data collection, offloading processing tasks like pre-processing, feature extraction, and ML modelling to cloud-based GPUs (through WiFi) [3], [19], desktop GPUs [20], mobile devices [1] or IoT devices [21]. This category of approaches can lead to high latency during signal transmission or raise privacy concerns. To address these challenges, our focus is on comprehensive WED for on-MCU computation, developing efficient and lightweight ML models suitable for limited-resource environments.

Efficient uncertainty estimation. Some effort has been devoted to achieving efficient uncertainty estimation, such as regulating the neural network weights to simulate BNNs [22]. Another stream of studies focuses on expensive and not deployable operations on MCUs like flow [23], spectral normalization [24], and stochastic Convolutional layers [9]. Despite their success in improving computation efficiency, their accuracy still either performs four times worse than the state-of-the-art (SOTA) method of deep ensembles [7] or require customized operators and libraries that are currently unavailable on MCUs. As an alternative to using ensembles, knowledge distillation [25] has been proposed as a means of training a single model. However, knowledge distillation typically requires out-of-distribution (OOD) data, which is often difficult to obtain for real-world applications. Compared to existing work, our study is the first to propose an efficient model for uncertainty quantification on MCUs.

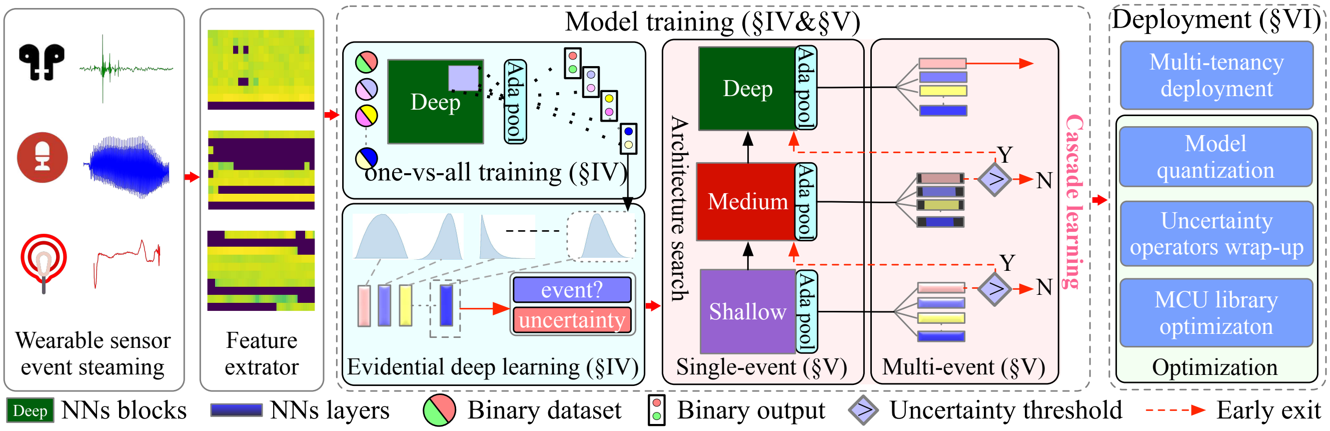


Fig. 2: System overview.

III. UR2M System overview

UR2M includes two stages: **model training** (§IV-§V) and **deployment** (§VI) as shown in Figure 2. During the **training stage**, there are three objectives: (1) EDL for efficient uncertainty quantification, (2) Cascade ML learning which includes single-event (intra-event) detection via early exits, and multi-event (inter-event) detection via feature sharing and multi-heads. During the **deployment stage**, we first carry out (1) multi-tenancy deployment [26], allowing multiple ML models (referred to as "tenants") to efficiently and dynamically share the same memory space among intra-event models. We then further focus on (2) optimizing the model and the MCU library.

In detail, wearable sensors first capture event streaming signals. Features are then extracted for different signals, such as Mel-frequency cepstral coefficients (MFCC) for the audio signals. Following this, evidential modeling via EDL and one-vs-all training **(§IV)** are applied to obtain reliable WED predictions and estimate uncertainty. Within the EDL framework, we specifically designed a cascade learning architecture **(§V)** for single-event detection, which divides the network layers into shallow, medium, and deep levels to enable intra-event sharing (sharing shallower layers and inferring with early exits within an event model) and process samples at different levels of recognition difficulty. Further, we propose inter-event sharing (sharing entire layers for feature extraction) for multi-event detection. In addition to the modeling, we further carry out efficiency improvements **(§VI)** via model architecture search (during model training), quantization, uncertainty operator wrap-up, and MCU library optimizations.

IV. Efficient Uncertainty Quantification

In this Section, we propose a highly efficient EDL model tailored for event detection on MCUs. This model is optimized to adhere to the constraints of MCUs, employing distributions to achieve accurate uncertainty quantification in real-time scenarios through a single forward pass.

A. Evidential Deep Learning

For a given input x^i, EDL generates a Dirichlet distribution $Dir(\boldsymbol{\alpha}^i)$, where $\boldsymbol{\alpha}^i = [\alpha_1^i, \alpha_2^i, ..., \alpha_C^i]$ denotes the concentration parameters of the distribution (dense distribution means high evidence and low uncertainty) [22]. Being a conjugate prior to the categorical distribution, the Dirichlet distribution enables EDL to determine the belief mass $\boldsymbol{b}^i = [b_1^i, b_2^i, ..., b_C^i]$ correlating directly with uncertainty. A higher belief mass indicates a higher confidence in the prediction, whereas a lower belief mass suggests the presence of uncertainty. Formally,

$$\boldsymbol{b}^i = (\boldsymbol{\alpha}^i - 1)/S^i, \tag{1}$$

where $S^i = \sum_{c=1}^{C} \alpha_c^i$ is the Dirichlet strength. From $\boldsymbol{\alpha}^i$ and $\boldsymbol{b}^i$, we can further infer the categorical prediction $\hat{y}^i$ and the associated uncertainty u^i as:

$$\hat{y}^i = \arg\max_c [\alpha^i / S^i], \quad u^i = 1 - \sum_{c=1}^{C} b_c^i \tag{2}$$

Before the training process, acknowledging our initial state of complete uncertainty about the outputs (i.e., uncertainty u^i is set to 1), we initialize $\boldsymbol{\alpha}^i$ with $[1, 1, 1]$, corresponding to $\boldsymbol{b}^i = [0, 0, 0]$ according to Eq. 1 and Eq. 2. To refine the model, we employ a loss function defined as:

$$\min_\theta \mathcal{L} = \frac{1}{N} \sum_i^N CE(\alpha_c^i / S^i, y^i) - \lambda \cdot H(Dir(\boldsymbol{\alpha}^i)) \tag{3}$$

where CE denotes the cross-entropy loss, and H represents the entropy of a Dirichlet distribution parameterized by $\boldsymbol{\alpha}^i$. The first term of the loss function aims to maximize classification accuracy, while the second term controls the output distribution to avoid overconfidence. The hyperparameter λ plays a crucial role in balancing these two terms.

Finally, this procedure will lead to a predicted $\boldsymbol{\alpha}^i$ for each sample which is used to infer the categorical outcome and the associated uncertainty (e.g., $u = 1 - \sum b^i$).

B. Efficient Evidential Modeling for Event Detection on MCUs

Implementing the EDL discussed in §IV-A for WED requires deploying multiple models and performing a series of inferences to detect various events, which significantly challenges the limited computational resources of MCUs. To mitigate this, we propose an efficient EDL modeling for WED, along with related training and optimization techniques designed to infer multiple events concurrently.

Efficient EDL Modeling for WED. WED is designed to identify an event signal coming from a wearable device. In ML/DL, this objective is defined as a binary classification task over a given duration/period of sensor data. For each binary classifier that detects classes of the event c, the outputs of EDL include the binomial belief mass, which can be used to infer the uncertainty of the WED prediction, i.e., how confident it is to be classified as positive (i.e., an event happening) or negative (i.e., an event not happening).

Given the binary nature of our EDL framework (positive vs negative), we adopt a Beta distribution (a special case of the Dirichlet distribution) to model the event probability. Specifically, a Beta distribution is characterized by two parameters α_c^i and β_c^i such that

$$\begin{aligned} \mathrm{P}(p_c^i \mid x^i; \theta_c) &= \mathrm{Beta}(p_c^i \mid \alpha_c^i, \beta_c^i) \\ &= \frac{1}{B(\alpha_c^i, \beta_c^i)} p^{\alpha_c^i - 1}(1-p)^{\beta_c^i - 1}, \end{aligned} \tag{4}$$

where $\mathrm{P}\left(p_c^i \mid x^i; \theta_c\right)$ denotes the probability distribution of the event given the sensor sample x^i, with both α_c^i, and β_c^i being greater than zero. $B(\alpha_c^i, \beta_c^i) = \Gamma(\alpha_c^i)\Gamma(\beta_c^i)/\Gamma(\alpha_c^i + \beta_c^i)$ is the Beta function, $\Gamma(\cdot)$ is the gamma function, and $p_c^i \neq 0$. Applying the mapping rule in Eq. 2, the prediction and uncertainty u for each sample i are derived via a NN:

$$b_1^i = \frac{\alpha_c^i - 1}{\alpha_c^i + \beta_c^i}, \; b_2^i = \frac{\beta_c^i - 1}{\alpha_c^i + \beta_c^i} \tag{5}$$

$$u^i = 2/(\alpha_c^i + \beta_c^i) \tag{6}$$

where b_1 represents the probability of a positive prediction while b_2 denotes that of a negative prediction.

One-versus-all classifiers. To obtain the parameters of α_c^i and β_c^i in EDL across multiple events, we adopt the one-versus-all (OVA) classifier, where each classifier distinguishes a specific event from all others, leading to C binary classifiers (i.e., heads). Specifically, in multi-event WED, we split the entire training dataset into C independent datasets with binary labels (i.e., event c vs. non-event c for $c \in [1, C]$). For each event, we then develop a model to learn a set of mapping functions $h_c(x^i; \theta_c)$, where x^i represents the input signal, and θ_c are the model weights. The outputs of the mapping functions yield the parameters α_c^i and β_c^i in the Beta distribution, computed as:

$$\alpha_c^i, \beta_c^i = h_c\left(x^i; \theta_c\right) \tag{7}$$

From this, we can deduce binomial decisions, with b_1^i denoting a positive prediction (i.e., event happening), and b_2^i representing a negative prediction (i.e., event not happening). Subsequently, these mapping functions are optimized jointly through an OVA training [27]. With this joint training of a shared EDL model, there is no need to deploy separate models on MCUs, thereby significantly reducing memory costs.

In contrast to traditional softmax-based deep learning approaches, which force the Neural Networks (NNs) to predict a point estimation, we can replace the softmax layer of the neural network with a ReLU layer (or an exponential function but softplus is not available in the MCU library). This adjustment ensures that the outputs remain non-negative, aligning with the positive α^i and enabling the NNs to predict distributions for each event task.

C. *Uncertainty-aware training and optimization*

Focusing on the training and optimization of the EDL framework for the proposed multi-event WED, we draw inspiration from Eq. 3 and propose using the binary cross entropy and Beta loss for each binary classifier of event c as:

$$\min_{\theta} \mathcal{L} = \frac{1}{N} \sum_{i}^{N} BCE\left(\psi_c^i / S_c^i, y_c^i\right) - \lambda \cdot H\left(B\left(\psi_c^i\right)\right) \tag{8}$$

where ψ_c^i symbolizes the Beta distribution parameters (α_c^i, β_c^i), BCE is the binary cross-entropy loss, H represents the entropy of a Beta distribution B parameterized by ψ_c^i and λ serves as a balancing weight between the cross-entropy loss and entropy of the Beta loss. For all C events, we collectively optimize all binary classifiers [28], enabling the model to perform inference with just a single forward pass.

V. Cascade learning

This section discusses designing efficient neural networks for UR2M. We explore the benefits of the early-exit strategy and architecture search method for single-event sharing on MCUs, reducing computational and memory costs. We also examine multiple-event sharing and detail the training pipeline using cascade learning, with all search and training *on the server*.

A. *Single-event Sharing*

For many DL tasks, some input samples, referred to as "easy" samples, can be effectively classified using shallower layers of the representation. This indicates that these shallower representations can identify "easy" samples, thus avoiding extra computation, whereas more "difficult" samples require processing through deeper layers [29]. However, unlike edge GPUs, designing model sharing on MCUs is challenging given the limited computing power, memory, and library support.

Using Early-exits to Share Shallower Layers. We propose a nested architecture featuring three early exits (sub-networks), which include shallow, medium, and deep models designed for single-event (intra-event) sharing, as illustrated in Figure 2 for MCUs. Each sub-network is designed using identical blocks of neural network layers, inspired by efficient neural networks for edge devices [30]. Existing early-exit methods usually rely on accuracy as a criterion to prune model branches. However, uncertainty can act as a crucial indicator for reliable prediction: we propose using uncertainty as a metric to determine whether to exit at each sub-network. As demonstrated in Figure 2, uncertainty thresholds are applied at the output of both shallow and medium models to facilitate early exits for data with low uncertainty (i.e., reliable predictions), thereby saving on MCU overheads.

Uncertainty-aware Architecture Search. To find efficient neural networks that minimize MCU overhead, recent studies have shown that the number of operations (OPS) and channel sizes [14] are two crucial factors. Considering this, we propose

```
Algorithm 1: The Search and training of UR2M
Input: Channel $L$, OPS size $O$, $\mathcal{D}^{TRAIN}$, $\mathcal{D}^{TEST}$
Output: Event prediction $y$ and uncertainty $u$
Data: Training data $\mathcal{D}^{TRAIN}$
/* search single-event model */
1  best_backbone, best_score = False, 0
2  for $i$ in $L$ do
3      for $j$ in $O$ do
           // Train candidate NNs backbone ($\mathbf{b}_{ij}$)
4          NN←$\mathbf{b}_{ij}(\mathbf{W}_{ij}, L_i, O_j)$
5          accuracy ← NN($\mathcal{D}^{TRAIN}$)
6          tradeoff ← accuracy/OPS
7          if tradeoff < best_score then
8              best_NN, best_score = NN, tradeoff
9          return best_NN
/* train with cascade learning */
10 for $l = 0, 1, 2$ do
       // take each output as next exit's input
11     $u$, output ←($\mathbf{b}_l$ ($\mathbf{W}_l$), $\mathcal{D}^{TRAIN}$)
12     $\mathcal{D}^{TRAIN}$ ← output
13     if converge then
14         return $\mathbf{W}_l$
```

an effective yet straightforward architecture search method to identify optimal neural networks for the early-exit models (i.e., shallow, medium, and deep models) in single-event sharing.

Specifically, we employ the Depthwise block as the OPS to control model depth, as it serves as an ideal proxy for managing model latency on MCUs [14]. The structure of each block consists of 1×1 Convolutions, 3×3 Depthwise Convolutions, and 1×1 Convolutions. We design each block using a 2D convolutional layer to to effectively handle various input types and extract the initial features. Subsequently, we use a consistent padding strategy to control the depth of OPS, ensuring that the output of each block matches its input. Lastly, we incorporate a linear classifier in each block as the output layer for single-event detection.

To define the model search space for efficient architectures on edge devices, we configure channel sizes L (ranging from 32 to 512) and OPS sizes O (3 to 7), drawing from models like MobileNet [30], DSCNN [31] for mobile devices, and MicroNets [14] for MCUs. This leads to 60 potential configurations ($N = L \times O$), each comprising three sub-networks. Our objective is to identify the optimal configuration N^* that balances minimal OPS with maximal accuracy. As outlined in Algorithm 1 (Lines 1-9), the search process involves initially setting a best backbone and score (Line 1), iterating through combinations of channel and OPS sizes (Lines 2-3), and assessing candidate NNs based on accuracy and operational space trade-offs (Lines 5-8), to ultimately select the most efficient and accurate NN backbone (Line 9).

B. Multiple-event Sharing

For a C multi-event detection task, a common approach is to develop individual models to ensure reusability across different applications or use cases and to optimize efficiency for each model [12]. These models can occupy C times the MCU memory and computation cost compared to a single-event model. However, some singular events may share similar characteristics, which can be captured by an identical network for feature extraction. For example, EEG signals are often used to detect alpha waves (event 1) and beta waves (event 2) using two independent models, despite the fact that both waves describe brain activities and can share certain information.

Using Heads to Share Entire Backbone. We propose our multi-event detection models, which share three sub-networks (i.e., shallow, medium, and deep) and consist of $C * 3$ adaptive classifiers (cf. Figure 2). Compared to multi-class classification, our multi-event sharing framework allows for more flexibility in single-event detection, which is especially preferred on low-power MCUs to ensure efficiency and reusability across multiple applications. Specifically, as illustrated in Figure 2, for each shared shallow, medium, and deep backbone NNs, we design C independent classifiers to distinguish the C events. Each classifier is composed of an adaptive pooling layer and a linear layer. The adaptive pooling layer aims to adjust the different output sizes from the searched shallow, medium, and deep sub-networks to match the input size of the classifiers. We optimize all the classifiers in a multi-task learning paradigm.

Uncertainty-aware Cascade Learning. To train the aforementioned shallow, medium, and deep models for MCUs, we propose an uncertainty-aware cascade model inspired by deep cascade learning for training our early-exit models. As illustrated in Algorithm 1 (Lines 10-14), we employ three optimizers for the three exits, with each exit representing one-third of the model layers. Initially, we train the first one-third of the layers in the searched backbone model and then utilize its output to train the second exit. Finally, we optimize the third exit.

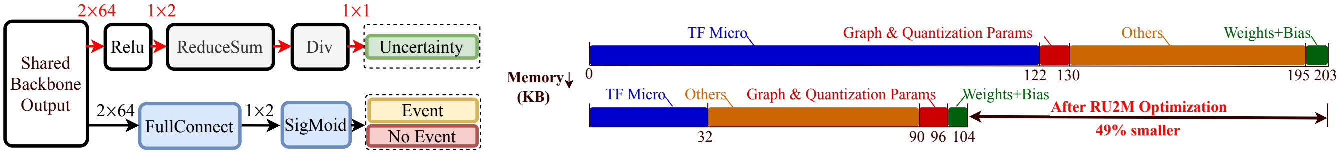
During each exit, we apply a single-layer linear layer (referred to as a head) for each event, which takes input maps of the output dimensions of the early-exits. Each early exit produces two outputs: the prediction and the uncertainty. We optimize all sub-networks concurrently on the server.

Our design is supported by the MCU libraries of Tensorflow Lite Micro (TFLM) in terms of multi-tenancy (e.g., enabling model deployment in a cascade manner) and memory planner (e.g., reusing the same operator's memory). This coherence can significantly reduce the overheads compared to the conventional multi-event detection models. Overall, our approach aims to optimize the performance of the models while accounting for uncertainty and providing early exits for faster inference.

VI. Implementation

A. System Implementation

Hardware. The training stage of our system is implemented and tested on a Linux server equipped with an Intel Xeon Gold 5218 CPU and NVIDIA Quadro RTX 8000 GPU. The shared backbone and multiple heads are pre-trained during this stage. Afterwards, in the deployment stage, we deploy the shared backbone and heads on two MCUs. The first one is



(a) Uncertainty classifier in TFLM format

(b) MCUs library optimization

Fig. 3: Deployment stage. (a) Uncertainty deployment on MCU based on multiple operators to calculate uncertainty and classification results. (b) MCU library space before optimization (top) and after optimization (bottom).

the STM32F446ZE (or F446ZE), which has an ARM Cortex M4 processor with 128 KB of SRAM and 512 KB of eFlash. The other one is the STM32H747XI (or 747XI), featuring a dual-core processor (ARM Cortex M4 and M7) with 1 MB of SRAM and 2 MB of eFlash. Our evaluation only utilizes one core (ARM Cortex M7) since MCUs are typically equipped with only one CPU core. This setup limits the usage space of SRAM and eFlash to 512 KB and 1 MB, respectively.

We developed and assessed our system's training stage using PyTorch 1.8, and tested various baselines on a Linux server. The evidential uncertainty module is implemented with Python and NumPy. We adopted TensorFlow Lite Micro (TFLM) [26] for MCU deployment due to its portability, ease of use, and support for numerous neural network layers and optimized kernels. UR2M's deployment stage and online optimization scheme are developed in C++ on two MCUs (ARM Cortex M4 and M7). To deploy a PyTorch model on MCUs, we convert it to TensorFlow Lite (TF Lite) using ONNX representation and the TF Lite converter. The model is run on MCUs using TFLM and Mbed OS. Additionally, the CMSIS-DSP software library processes raw signals to generate model inputs (e.g., MFCC features), and the CMSIS-NN kernels in TFLM facilitate efficient neural network operations on MCUs.

Multi-tenancy Deployment. To facilitate multi-event sharing on MCUs with limited memory, we develop a multi-tenancy deployment for early-exit models using TFLM. UR2M utilizes multiple model interpreters to allocate memory from a unified space, ensuring efficient model operation. During evaluation, this deployment strategy is applied to all baselines and the UR2M model. For example, Deep Ensembles have five models, potentially using 5× eFlash space. However, with optimization, it only consumes 2× more SRAM (cf. §VIII-C) due to multi-tenancy deployment.

B. Uncertainty Operator Implementation

To capture the uncertainty at inference time on MCUs, we only use TFLM-supported operations. First, we utilize a ReLU operator to regulate the distribution of the output as non-negatives. Then, based on these outputs, we follow Eq.6 to generate uncertainties. Specifically, calculating uncertainty first requires the sum of reduced dimensions. Although the *reduced_sum* operator is supported, it is not available for TFLM. To solve this, we use a *squeeze* operator to reduce the output dimensions, followed by a *sum* operator. Finally, we apply a *divide* operator to generate the uncertainty. We wrap the above-mentioned operators within the model and implement them in the TFLM library to save the overhead of uncertainty prediction. The overall uncertainty implementation is shown in Figure 3a.

C. MCU Library Optimization

Unlike mobile devices' memory architecture that employs large off-chip main memory (e.g., DRAM), MCUs consist of only small-sized on-chip memory (e.g., SRAM and eFlash) (cf. Figure 1). To understand the memory requirements of our model to fit in MCUs, we first compute the memory usage of UR2M. For a searched shallow model with 8-bit int quantization, we observe that TFLM requires 79 KB of SRAM and 203 KB of eFlash, which falls within the tight memory budgets of many MCUs, for example, 64 KB of SRAM and 128 KB of eFlash of STM32F205VB as described in Figure 1. In particular, on SRAM, the memory usage includes intermediate tensors (30 KB), persistent buffers (3 KB), runtime overhead of the TFLM interpreter (6 KB), and MBed OS and other libraries (10 KB). Additionally, Figure 3b top shows the on-chip eFlash architecture of an F446ZE MCU and how TFLM allocates memory space to run a shallow model on an MCU.

Note that since we only conduct 8-bit post-quantization, we only observe a maximum of 1% performance drop between the pre-and post-quantization stages among all methods.

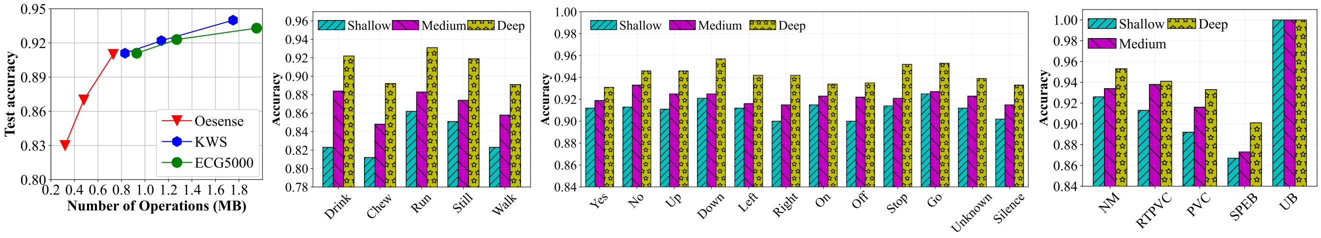
Given the limited memory space for searching the optimal model parameters, we propose optimizing the TFLM library. First, we removed all operation-related files that did not impact our backbone. Then, we reordered the operations files based on our backbone structure. As shown at the bottom of Figure 3b, our MCU library significantly optimized the TFLM interpreter's runtime overhead, reducing it from 122 KB to 32 KB (3.8× smaller). Moreover, the graph definition was reduced by 2 KB, from 8 KB to 6 KB, in the eFlash memory. After the optimization, a total of 104 KB of memory is used, which can now fit into the STM32F205VB and many other MCUs. Overall, UR2M optimizes 49% of eFlash memory compared to the baseline TFLM library.

Note that during the evaluation, we applied the same MCU library optimization strategy to all baselines as well as the UR2M model.

VII. Evaluation Settings

A. Evaluated Datasets

Our target application scenarios are focused on WED applications. Specifically, we evaluate three wearable datasets, including in-ear activity recognition [32], audio event keyword spotting [33], and heart disorder event detection [34]. We



(a) Model sizes vs. accuracy (b) Performance on Oesense (c) Performance on KWS (d) Performance on ECG5000

Fig. 4: Model sizes vs. Accuracy and early exit result for single events. Note that the ECG5000 UB event has only one test sample.

experiment with these three datasets, each featuring different data modalities that suit UR2M settings. For imbalanced datasets, we use SMOTE [35] to upsample the training data.

In-ear Dataset. Oesense [32] contains an in-ear audio dataset for activity recognition (including five events: “walk”, “run”, “still”, “drink”, and “chew”) among 31 subjects. For preprocessing, we first segment the original audio into one-second segments and set the sampling rate at 4 kHz. Then, we extract the 2-D MFCC features for each segment. 10 MFCC features are then obtained from an audio frame with a length of 80 ms and a stride of 40 ms, yielding an input dimension of 1×10×21. After preprocessing each event, we obtained 40,064 training samples (90%) and 4,452 test samples (10%) for all five activities.

KWS Dataset. The Keywords Spotting (KWS) V2 [33] dataset contains 105,829 utterances from 2,618 speakers. There are 35 words split into 12 classes, including ten keyword spotting classes and an ’unknown’ class (remaining 24 words). For preprocessing, we first constrained all event samples to one second by segmentation or zero-padding and set the sampling rate at 16 kHz. Then we extracted MFCC features using 640 FFT points and 320 points of hop length. We obtained 10 MFCC features from an audio frame with a length of 40ms and a sliding window of 20ms, yielding the input dimension of 1×10×51. After preprocessing, we obtained 92,502 total event training samples (90%) and 10,278 test samples (10%).

ECG5000 Dataset. The ECG5000 dataset [34] is a 20-hour long one-channel ECG dataset that contains 92,584 heartbeats, including five different types of heart events: Normal (NM) (58.4%), R-on-T Premature Ventricular Contraction (RTPVC) (35.3%), Premature Ventricular Contraction (PVC) (3.9%), Supra-ventricular Premature or Ectopic Beat (SPEB) (2%), and Unclassified Beat (UB) (0.5%). For preprocessing, we resample the input duration of 0.56s with 140 samples into 560 samples. Then we reshape the input into 10 channels, yielding the input dimension of 1×10×56. After the preprocessing, we obtained 4,500 total event training samples (90%) and 500 (10%) test samples. Note that UB has only one test sample.

B. Uncertainty Metrics

We compare UR2M using three important uncertainty metrics: Brier score, Negative Log-Likelihood (NLL), and Expected Calibration Error (ECE), to examine the uncertainty estimation performance.

C. Uncertainty Quantification Baselines

We evaluate the proposed method by comparing it to three baseline uncertainty solutions: the traditional softmax-based models, deep ensembles and data augmentation. It is important to note that MCDP [10] is not available for the MCUs library TFLM because it stores models as binary files that cannot be modified. Moreover, its computational costs are similar to or greater than deep ensembles, while its uncertainty performance is lower than that of deep ensembles [9].

Vanilla EDL. Vanilla EDL [22] is the state-of-the-art (SOTA) model to *efficiently* quantify uncertainty and can be implemented on MCUs.

Deep Ensembles. Deep ensembles approach (denoted as D(Softmax)+Ense) [7] is the SOTA model to *accurately* quantify uncertainty estimation, which typically ensembles N deterministic Softmax models with random weight initializations. We use $N = 5$ which is widely adopted in recent efficient studies [36].

Data Augmentation. Test time data augmentation (denoted as D(Softmax)+InAug) [37] is a *memory-efficient* uncertainty quantification method generating multiple test samples by applying data augmentation techniques through a single model. We utilize five augmented samples, incorporating Jittering, with a mean ε of 0 and a standard deviation σ of 0.03, which are added to the test data.

VIII. Results

This section will discuss the results and answer the following questions: (1) How efficient is UR2M for typical MCUs? (2) How robust is UR2M compared with traditional point prediction models?

A. Performance of Event Detection

Utilizing the Adam optimizer with a learning rate of $1e{-}3$, a 32 batch size, and an early stopping of 5 epochs, we train our networks, showcased in Figure 4a and Figures 4b- 4d. While system accuracy generally increases with OPS across all datasets, significant increases in overhead do not invariably equate to notable accuracy improvements, as observed in the KWS and ECG5000 datasets. For instance, a shallow Oesense

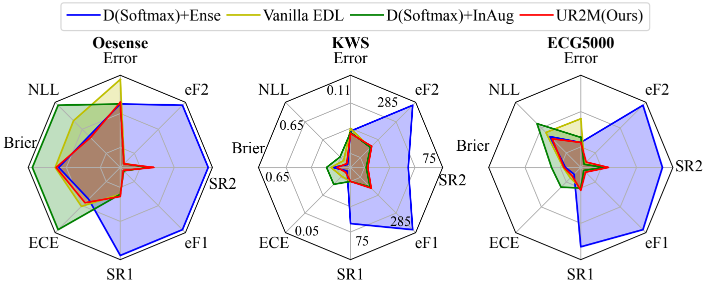


Fig. 5: Comparing Vanilla EDL, data augmentation, deep ensembles (SOTA), and UR2M using uncertainty, error rate, and memory usage metrics across three datasets. eF1 and SR1 refer to the eFlash and SRAM usage of H747XI, while eF2 and SR2 refer to those of F464ZE, respectively. For all metrics, lower values are preferred.

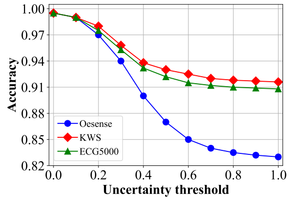


Fig. 6: Uncertainty impact.

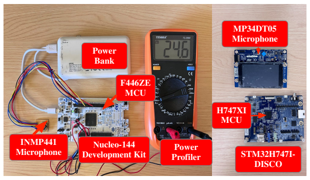


Fig. 7: End-to-end deployment.

model (accuracy: 0.83, parameters: 0.38 MB) contrasts with the medium and deep models, which respectively present 0.87/0.58 MB and 0.91/0.76 MB in accuracy/parameters. The 2% accuracy enhancement when transitioning from medium to deep models incurs a 31% overhead spike. Similarly, for ECG5000, a 1% accuracy improvement requires doubling the model sizes. Shallow models across all datasets exhibit proficient performance (e.g., >80%) with minimized model size, hinting that UR2M could deliver effective performance with modest overheads.

Regarding the channel sizes, our searched model yields the output shape for each OPS as [5,11] for Oesense, [5, 26] for KWS, and [5, 29] for ECG5000, respectively. Figure 4 further illustrates the UR2M performance for single event detection using shallow, medium and deep network structures.

Based on Figure 5, we can observe that UR2M's uncertainty metrics are better than Data Augmentation (D(Softmax)+InAug) baseline across all three datasets, with up to 22% lower NLL scores (0.65 to 0.53). This improvement indicates that the proposed method produces better-calibrated models that are less prone to overconfidence errors. Compared to the D(Softmax)+Ense model, UR2M achieves similar performance in terms of both uncertainty estimation and prediction accuracy. For instance, UR2M outperforms D(Softmax)+Ense by 8.0% in terms of Brier score for KWS, and achieves 1.7% and 2.4% relative improvements in NLL for Oesense and ECG5000, respectively.

Notably, UR2M achieves these results while using up to only half of the memory, much less energy, and latency required by SOTA method deep ensembles (cf. Figure 5 and §VIII-C), demonstrating the computational efficiency of UR2M without compromising uncertainty estimation.

B. Impact of different Uncertainty Thresholds

Users can decide on the uncertainty threshold according to their specific applications. For example, in healthcare applications (e.g., heart attack detection), we prefer a low uncertainty (e.g., u=0.05) for detected heart attacks to avoid disastrous consequences. This tradeoff is depicted in Figure 6. In other scenarios (e.g., running detection), a higher uncertainty threshold can be tolerated to save battery life by exiting through shallow layers. Similarly, it is observed that increasing the threshold gradually reduces latency across all three datasets when evaluated on the F446ZE and H747XI MCUs. With a higher uncertainty threshold, more samples are filtered out by the shallow and medium sub-networks, and fewer samples pass through deep models, leading to reduced latency. This indicates that selecting different uncertainty thresholds allows users to obtain a personalized model, increasing the usability of UR2M.

In sum, our model design, which allows users to define the threshold, can help determine the optimized threshold to balance the tradeoff, thereby achieving personalized models.

C. End-to-end System Efficiency

Following the optimization of all baselines and UR2M using techniques including multi-tenancy deployment, model quantization, and MCU library optimization, we evaluate their runtime efficiency during deployment on MCUs (Figure 7). Our evaluation encompasses the entire system, including signal acquisition, feature extraction, and memory usage in terms of SRAM and eFlash required for model execution. We conducted experiments with various datasets and two typical resource-constrained MCUs, the F446ZE and H747XI. Although the focus is primarily on the ECG5000 dataset due to page limits, note that consistent outcomes were observed across all three datasets.

Model Inference Memory Footprint. Based on our implementation, UR2M consumes only 49 KB and 51 KB of SRAM (38.5% and 9.9% of the total SRAM of F446ZE and H747XI, respectively) as shown in Figure 5. Additionally, as shown in Figure 5, UR2M requires 142 KB and 145 KB of eFlash (27.7% and 14.1% of the total eFlash of F446ZE and H747XI, respectively). These results demonstrate that UR2M consumes only a small portion of the limited resources of MCUs, leaving enough resources for other applications to be supported simultaneously. Furthermore, UR2M requires only 66-67% of SRAM (49 KB vs. 75 KB for F446ZE and 51 KB vs. 75 KB for H747XI) and 51% of eFlash (142 KB vs. 280 KB for F446ZE and 145 KB vs. 283 KB for H747XI) compared to the deep ensembles baseline.

Signal Acquisition Overheads. To evaluate signal acquisition overheads for the F446ZE MCU, we employ an INMP441 MEMS microphone. For the H747XI, we use the MP34DT05-A built-in microphone on the H747I-DISCO evaluation board (Figure 7). We assess energy consumption and memory usage as key factors. Energy consumption (J) is computed as the product of time/latency (t) and power (W). Power is determined from input voltage (V) and current measurements (A), conducted

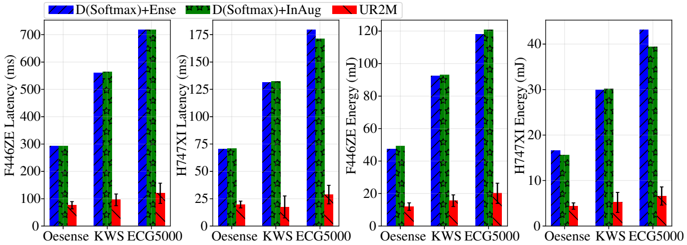


Fig. 8: Comparison of latency and energy consumption of uncertainty-aware methods on two MCUs.

with a Fluke 87V digital multimeter. For the F446ZE, we record a power consumption of 24.6 mA at 3.3V, resulting in 81.18 mW for one second of audio signal acquisition. Memory-wise, it uses 4KB of SRAM and 32KB of eflash. In contrast, the H747XI consumes 31.6 mA at 3.3V, totaling 104.28 mW in power. It utilizes 29KB of SRAM and 66KB of eflash. Overall, signal acquisition overheads for these two MCUs are minimal.

Feature Extraction Overheads. The feature extraction step for both UR2M and the baselines is the same, using MFCC features as inputs. The extraction process is fast, taking only 4.505 ms and 10.913 ms per extraction for the H747XI across two datasets, indicating minimal overhead.

Model Inference Latency. Using the MBed Timer API to measure latency on MCUs, Figure 8 illustrates UR2M's and baseline inference results across three datasets and two MCUs. With uncertainty thresholds (u) ranging from 1 to 0, UR2Mpresents latencies from lowest to highest, respectively. While baseline approaches, like deep ensemble, yield reliable uncertainty estimations, they exhibit high inference latencies of 717.2-717.4 ms on F446ZE and 171.1-179.3 ms on H747XI per sample. Conversely, UR2Mensures both reliable uncertainty and minimized latency, cutting inference latencies up to 864% (83.0 ms vs. 717.2 ms) on F446ZE and 835% (20.2 ms vs. 171.1 ms) on H747XI. Moreover, UR2Menhances latency by approximately 456% against other baselines, even without uncertainty filtering.

Model Inference Energy Consumption. Similar to the latency results, UR2M significantly reduces energy consumption compared to the baselines, as shown in Figure 8. For example, UR2M decreases energy consumption by up to 834% (116.0 mJ vs. 13.9 mJ) on F446ZE and 857% (39.4 mJ vs. 4.6 mJ) on H747XI when compared to the best-performing benchmark uncertainty-aware baselines. Also, we observe that UR2M achieves around 450% energy improvement compared to the baselines without uncertainty filtering.

D. Robustness Against Signal Uncertainties

We evaluate UR2M in the context of two types of signal uncertainties: signal missing (replace as zero) and noise (gaussian noise). Due to page limitations, we compare our method with traditional softmax-based NNs having the same model structure. As demonstrated in Figure 9, for a correct event signal "Chew", the absence of signal and the presence of random noise can lead softmax-based NNs to predict incorrectly. In contrast, UR2M can accurately predict most corrupted signals. When predictions are incorrect, UR2M also exhibits high uncertainty (e.g., u=1.0), which could be used for alerting the system to potential misclassifications or triggering additional validation steps.

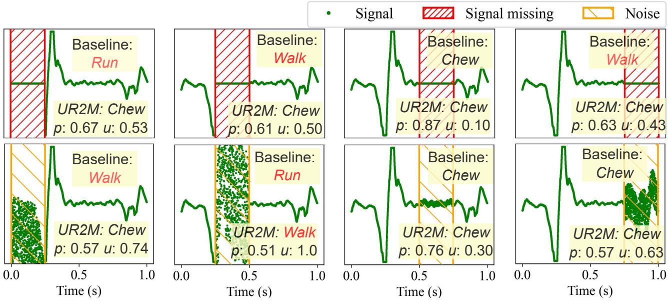


Fig. 9: Uncertainty estimation towards signal missing and noise. Labels in red indicate wrong predictions.

IX. Discussion

In this section, we discuss several possible future directions for our work.

Generalizing UR2M to other sensors and higher-end MCUs. Ideally, UR2M could be generalized to any wearable sensors driven by MCUs. However, sensor signal complexity and limited MCU memory size pose limitations. More complex signals usually require larger model sizes, challenging the deployment on the constrained memory of MCUs. Fortunately, recent work [12] shows that by investigating compressive sensing, key patterns of primitives in signals can be compressed and extracted, which indicates it can reduce the model size to save system overhead. Therefore, we will study how compressive sensing combined with UR2M could further reduce system overhead to generalize to ultra low-end MCUs. We envision our method could also benefit higher-end MCUs, e.g. STM32F4, which has 1MB flash and 192KB SRAM. Since less memory is required, higher-end MCUs could experience improvements in latency and energy efficiency.

Impact of UR2M on future WED systems. Our work has illustrated that uncertainty is a key criterion to ensure reliable prediction in WED systems. Therefore, an important and urgent question is how to define uncertainty tolerance thresholds for specific applications. Fortunately, for healthcare applications, we can design this criterion through a doctor-in-the-loop strategy to select the optimal threshold.

X. Conclusion

In this paper, we have proposed UR2M, a resource and uncertainty-aware framework which can efficiently and reliably enable wearable event detection and related uncertainty on MCUs. By exploiting evidential uncertainty theory, cascade learning, and system optimization, UR2M significantly improves energy and memory efficiency for MCUs without sacrificing accuracy, enabling real-time and reliable event detection.

XI. Acknowledgment

This work is supported by ERC through Project 833296 (EAR), and Nokia Bell Labs through a donation.

References

[1] Arash Alavi, Gireesh K Bogu, Meng Wang, Ekanath Srihari Rangan, Andrew W Brooks, Qiwen Wang, Emily Higgs, Alessandra Celli, Tejaswini Mishra, Ahmed A Metwally, et al. Real-time alerting system for covid-19 and other stress events using wearable data. *Nature medicine*, 28(1):175–184, 2022.

[2] Christian Holz and Edward J. Wang. Glabella: Continuously sensing blood pressure behavior using an unobtrusive wearable device. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3), sep 2017.

[3] Yuezhou Zhang, Zhicheng Yang, Zhengbo Zhang, Peiyao Li, Desen Cao, Xiaoli Liu, Jiewen Zheng, Qian Yuan, and Jianli Pan. Breathing disorder detection using wearable electrocardiogram and oxygen saturation. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 313–314, 2018.

[4] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

[5] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

[6] Gustavo Carneiro, Leonardo Zorron Cheng Tao Pu, Rajvinder Singh, and Alastair Burt. Deep learning uncertainty and confidence calibration for the five-class polyp classification from colonoscopy. *Medical Image Analysis*, 62:101653, 2020.

[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[8] Yucheng Wang, Mengmeng Gu, Mingyuan Zhou, and Xiaoning Qian. Attention-based deep bayesian counting for ai-augmented agriculture. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 1109–1115, 2022.

[9] Lorena Qendro, Jagmohan Chauhan, Alberto Gil CP Ramos, and Cecilia Mascolo. The benefit of the doubt: Uncertainty aware sensing for edge computing platforms. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 214–227. IEEE, 2021.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[11] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *Advances in Neural Information Processing Systems*, 33:7498–7512, 2020.

[12] Nhat Pham, Hong Jia, Minh Tran, Tuan Dinh, Nam Bui, Young Kwon, Dong Ma, Phuc Nguyen, Cecilia Mascolo, and Tam Vu. Pros: an efficient pattern-driven compressive sensing framework for low-power biopotential-based wearables with on-chip intelligence. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 661–675, 2022.

[13] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[14] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems*, 3:517–532, 2021.

[15] Edgar Liberis, Łukasz Dudziak, and Nicholas D. Lane. unas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, EuroMLSys '21, page 70–79, New York, NY, USA, 2021. Association for Computing Machinery.

[16] Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frameexit: Conditional early exiting for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15608–15618, 2021.

[17] Erika Bondareva, Elín Rós Hauksdóttir, and Cecilia Mascolo. Earables for detection of bruxism: a feasibility study. In *Adjunct Proceedings of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2021 ACM International Symposium on Wearable Computers*, pages 146–151, 2021.

[18] Dariusz Wójcik, Tomasz Rymarczyk, Michał Oleszek, Łukasz Maciura, and Piotr Bednarczuk. Diagnosing cardiovascular diseases with machine learning on body surface potential mapping data. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 379–381, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019.

[20] Tuochao Chen, Yaxuan Li, Songyun Tao, Hyunchul Lim, Mose Sakashita, Ruidong Zhang, Francois Guimbretiere, and Cheng Zhang. Neckface: Continuously tracking full facial expressions on neck-mounted wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(2), jun 2021.

[21] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2020.

[22] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. *arXiv preprint arXiv:1806.01768*, 2018.

[23] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31, 2018.

[24] Jishnu Mukhoti, Andreas Kirsch, Joost van Amersfoort, Philip HS Torr, and Yarin Gal. Deep deterministic uncertainty: A simple baseline. *arXiv e-prints*, pages arXiv–2102, 2021.

[25] Andrey Malinin, Bruno Mlodozeniec, and Mark Gales. Ensemble distribution distillation. *arXiv preprint arXiv:1905.00076*, 2019.

[26] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.

[27] Shreyas Padhy, Zachary Nado, Jie Ren, Jeremiah Liu, Jasper Snoek, and Balaji Lakshminarayanan. Revisiting one-vs-all classifiers for predictive uncertainty and out-of-distribution detection in neural networks. *arXiv preprint arXiv:2007.05134*, 2020.

[28] Gianni Franchi, Andrei Bursuc, Emanuel Aldea, Séverine Dubuisson, and Isabelle Bloch. One versus all for deep neural network incertitude (ovnni) quantification. *arXiv preprint arXiv:2006.00954*, 2020.

[29] Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 235–244, 2020.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[31] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[32] Dong Ma, Andrea Ferlini, and Cecilia Mascolo. Oesense: employing occlusion effect for in-ear human sensing. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 175–187, 2021.

[33] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[34] Yanping Chen, Yuan Hao, Thanawin Rakthanmanon, Jesin Zakaria, Bing Hu, and Eamonn Keogh. A general framework for never-ending learning from time series streams. *Data mining and knowledge discovery*, 29(6):1622–1664, 2015.

[35] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[36] Lorena Qendro, Alexander Campbell, Pietro Lio, and Cecilia Mascolo. Early exit ensembles for uncertainty quantification. In *Machine Learning for Health*, pages 181–195. PMLR, 2021.

[37] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.