ePerceptive—Energy Reactive Embedded Intelligence for Batteryless Sensors

Alessandro Montanari
Nokia Bell Labs, UK
alessandro.montanari@nokia-bell-labs.com

Manuja Sharma*
University of Washington, USA
manuja21@uw.edu

Dainius Jenkus*
Newcastle University, UK
d.jenkus1@newcastle.ac.uk

Mohammed Alloulah*
Nokia Bell Labs, UK
mohammed.alloulah@nokia-bell-labs.com

Lorena Qendro
University of Cambridge, UK
lorena.qendro@cl.cam.ac.uk

Fahim Kawsar
Nokia Bell Labs, UK and TU Delft, NL
fahim.kawsar@nokia-bell-labs.com

ABSTRACT

For long, we have studied tiny energy harvesters to liberate sensors from batteries. With remarkable progress in embedded deep learning, we are now re-imagining these sensors as intelligent compute nodes. Naturally, we are approaching a crossroad where sensor intelligence is meeting energy autonomy enabling maintenance-free swarm intelligence and unleashing a plethora of applications ranging from precision agriculture to ubiquitous asset tracking to infrastructure monitoring. One of the critical challenges, however, is to adapt intelligence fidelity in response to available energy to maximise the overall system availability. To this end, we present the design and implementation of **ePerceptive**: a novel framework for best-effort embedded intelligence, i.e., inference fidelity varies in proportion to the instantaneous energy supplied. ePerceptive operates on two core principles. First, it enables training a single deep neural network (DNN) to operate on multiple input resolutions without compromising accuracy or incurring memory overhead. Second, it modifies a DNN architecture by injecting multiple exits to guarantee valid, albeit lower-fidelity inferences in the event of energy interruption. The combination of these techniques offers a smooth adaptation between inference latency and recognition accuracy while matching the computational load to the available power budget. We report the manifestation of ePerceptive in designing batteryless cameras and microphones built with TI MSP430 MCU and off-the-shelf RF and solar energy harvesters. Our evaluation of these batteryless sensors with multiple vision and acoustic workloads suggest that the dynamic adaptation of ePerceptive can increase the inference throughput by up to 80% compared to a static baseline while ensuring a maximum accuracy drop of less than 6%.

CCS CONCEPTS

• **Software and its engineering → Embedded software**; • **Computing methodologies → Neural networks**; • **Hardware → Analysis and design of emerging devices and systems**.

KEYWORDS

Energy autonomous, embedded intelligence, batteryless devices.

ACM Reference Format:
Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive—Energy Reactive Embedded Intelligence for Batteryless Sensors. In *The 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3384419.3430782

1 INTRODUCTION

The long-sought vision of swarm intelligence is in our periphery. Resilient efforts in energy engineering and harvesting circuits have brought us closer to a graspable future in which, billions and trillions of tiny batteryless sensors will collect, process and communicate data about people, places and things [39]. In parallel, we are observing a steady transition of machine intelligence from on-cloud solutions to on-device (e.g., smartphones, smartwatches, etc.) solutions [1, 35]. With remarkable advancement in embedded deep learning and SoC accelerators, we are now pushing computation further down the system stack - bringing intelligence at the sensor level [18, 38]. The convergence of these two threads means, for the first time, we will be able to realise the "Deploy and Forget" vision whereby trillions of sensors will perceive and act in the environment around us and will uncover myriad of applications across verticals. For instance, in agriculture, every plant could be monitored for precision irrigation, ensuring high yield at minimum cost. In fleet and asset management, disposable tags could enable fine-grained tracking to minimise financial losses caused by missing inventories or to increase RoI achieved through superior customer satisfaction. The applications are endless including wild-life conservation [16], healthcare [10], energy-efficient buildings [2] and many more.

Batteryless sensors first harvest energy (e.g. from solar or RF) and buffer it in the storage and then, when sufficient energy is available, they activate to perform their task (i.e. sensing, compute and communication). Naturally, such a sensor is characterised by

*This work was done when these authors were affiliated with the Pervasive Systems Department at Nokia Bell Labs Cambridge.

SenSys '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-7590-0/20/11…$15.00
https://doi.org/10.1145/3384419.3430782

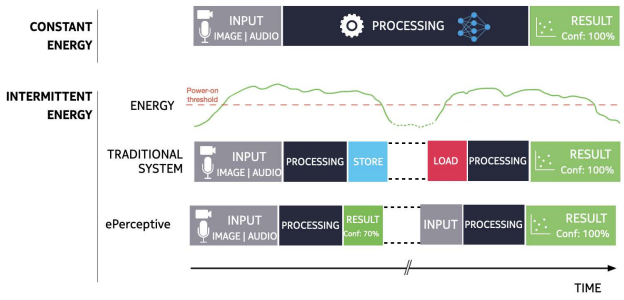


Fig. 1: ePerceptive enables batteryless sensors to dynamically adapt the inference quality in response to available energy by utilising DNNs that operate on multi-resolution inputs and provide inferences of varying fidelity with multiple exit branches.

high uncertainty, as the time at which it is awake and can sense the environment or process incoming data is highly unpredictable [33]. Given the challenging operating conditions, batteryless sensors are often confined to low-fidelity and low data-rate sensors, such as inertial measurement units or environmental sensors. Their data processing pipelines are also limited to simple operations like averaging or peak detection. Recent works on embedded machine learning have shown how accurate, on-device Deep Neural Networks (DNNs) could significantly improve the perception ability of ultra low-power devices and thereby reduce data transmission overhead enabling applications that can run for longer [18, 29, 38].

However, one of the critical challenges to bring embedded deep learning into batteryless sensors is to make the model's operational behaviour reactive to the uncertain energy supply of the sensors, to maximise the overall system availability. DNN models, typically monolithic, require invasive retraining, and in some instances architectural restructuring when their resource footprints need to be adjusted. Although there are several techniques for such model re-engineering [6, 25, 28, 32], the process is static and unable to flexibly adapt to fluctuating energy availability. Moreover, current DNN models, once retrained or restructured, can output an inference if and only if resource availability is above a requisite threshold. Such rigidity is, again, problematic for batteryless sensors as when available energy falls below this threshold, sensor nodes are unable to convert data into inferences.

Building on these observations, in this paper, we investigate a specific challenge, namely: *"Given that energy availability of a batteryless sensor varies over time, can we design an energy reactive data processing and inference pipeline that can offer smooth adaptation between compute latency and recognition accuracy while matching the computational load to the available energy budget?"*

To this end, we present the design and implementation of ePerceptive: a novel framework for best-effort inference—i.e. inference whose quality varies in proportion to the instantaneous energy supplied as illustrated in Figure 1. This is in contrast to traditional application-driven approaches, where the focus is on minimising the overall power consumption disregarding the instantaneous energy envelope. ePerceptive operates on two core mechanisms and is implemented on an extended version of the SONIC intermittent computing framework [18]:

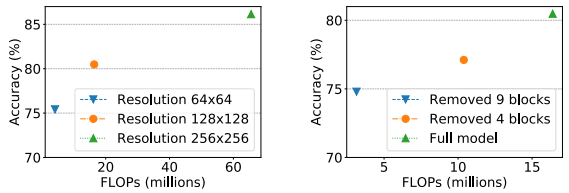
- The first technique exploits the fact that the computational complexity and thereby the energy footprints of DNNs is directly proportional to their input size, typically with increased accuracy when the input fidelity is higher. ePerceptive takes advantage of this principle and enables the training of a single model which operates accurately at different input resolutions without the overhead of storing multiple models in memory. This multi-resolution inference technique enables the selection of the input dimension, which ensures the required level of accuracy while keeping the energy necessary for the inference within the available limit, a key requirement for batteryless sensors (see § 4).
- The second technique borrows principles from anytime algorithms and early-exit DNNs [7, 41] and contextualise them in a batteryless setting. This technique enhances a DNN architecture through careful injection of multiple exits to enable valid inferences even in the event of energy interruption. This multi-exit inference mechanism enables a batteryless sensor to output lower-fidelity inferences at intermediate layers in addition to the final deepest output. The technique is introduced in § 5.
- ePerceptive leverages SONIC framework [18] to manifest these two adaptive techniques, however with sophisticated extensions for flexible and controlled grading of performance for batteryless sensors. We introduce a new model interpreter, purpose-built buffer management and a light-weight scheduler to enable SONIC to support multi-resolution and multi-exit inferences on batteryless sensors. Details of ePerceptive implementation are in § 6.

We have designed a batteryless camera and a batteryless microphone with ePerceptive, and evaluated them using two popular vision datasets, Visual Wake Words [12] (people detection) and Caltech Camera Traps [4, 5] (animal detection) and one acoustic dataset, Speech Commands v0.02 [43]. We implement ePerceptive with two embedded models, variations of MobileNetV1 [23, 48]. These workloads represent realistic but challenging learning tasks for ultra-low-power sensors in the wild. Our evaluation results suggest that the dynamic adaptation of ePerceptive can increase the inference throughput of these sensors by up to 80% compared to a static baseline while ensuring a maximum accuracy drop of less than 6%. Taken together these and the rest of our findings demonstrate that ePerceptive can adapt with grace to widely fluctuating and unknown energy operational conditions.

In what follows, we first reflect on the challenges of designing batteryless sensors with embedded intelligence. Then we present the architecture of ePerceptive followed by the detail description of its two fundamental principles. We then move to the implementation of ePerceptive and describe its hardware and software. Then we present the evaluation of ePerceptive with two batteryless sensors and multiple perception workloads. We then position our research concerning related work and conclude the paper.

2 BACKGROUND AND CHALLENGES

Energy harvesting systems, or batteryless sensors, have the potential to enable endless applications at scale with very low cost of deployment and maintenance [2, 10, 16]. However, they present significant challenges as they rely on harvesting unpredictable



(a) (b)

Fig. 2: Relationship between floating point operations (FLOPs) and accuracy when changing (a) input resolution and (b) number of layers. MobileNetv1 mdoels trained on the VWW dataset [12].

energy from the environment and demand radical changes in hardware [13, 14, 21] and software [22, 34, 47] design to ensure robustness to sudden power failures. Thanks to the maturity of recent hardware and software platforms, we are increasingly observing complex applications on these systems primarily aiming at detecting interesting events with high accuracy, and minimising communication cost. Indeed, recent works highlight how accurately detecting interesting events with Deep Neural Networks (DNN) plays a significant role in reducing communication overhead and increasing the utility of these systems [18, 38].

DNN models are resource hungry and, more importantly, not easily scalable at runtime. In fact, once trained, they produce inferences only when there are enough resources (i.e., energy) to execute the entire network. While intermittent-safe implementations can guarantee that an inference could span several power failures [18], they do not automatically enable existing models to produce a valid output that meets the instantaneous energy envelope. This implies that, when energy is scarce, a fixed-latency big model will waste energy completing an inference that could take significant time. On the other hand, if only a faster but less accurate model is used, the system cannot take advantage of excess available energy, and will always produce sub-optimal inferences.

When building a new DNN, there are three control parameters to scale its computational complexity: (1) the input resolution, (2) the number of layers and (3) the number of filters per layer [40]. Other techniques involve re-engineering an existing model after the initial training [6, 25, 28, 32]. However, without specialised treatment, these techniques do not solve the problem of adapting the models' complexity to the variable energy availability.

Figure 2 shows this point. When we train models at higher resolutions (Figure 2a), we obtain progressively more accurate models at the expense of significant computational complexity. Similarly, when we remove layers from the model, we observe drops in accuracy (Figure 2b). These, however, are individual models that need to be trained separately and, once trained, have a fixed computational requirement which cannot be changed at runtime to match a fluctuating energy envelope. A simple way to achieve such dynamism would be to store all models in memory and select the appropriate one at runtime based on the energy conditions. However, this is prohibitive for energy harvesting devices which have small memories, in the order of KBs (e.g. 256KB or 512KB). Techniques like weights quantisation [25], pruning [32] or layer decomposition [28] could alleviate this issue. Nevertheless, they suffer accuracy drops when heavy compression is applied and, moreover, add the overhead of having to train multiple models.

Table 1: Time and energy consumed by pre-processing operations on the TI MSP430FR5994 platform. l and s are the frame length and stride respectively, used for the MFCC computation.

Pre-Processing Operation	Time (ms)	Energy @ 2.2v (mJ)
QQVGA image acquisition	200	1.10
Microphone sampling (8kHz)	1000	3.96
Image resize	50	0.27
MFCC (l = 30ms, s = 10ms)	1280	5.13
MFCC (l = 30ms, s = 20ms)	640	2.56
MFCC (l = 30ms, s = 30ms)	430	1.72

Besides, different input resolutions and feature engineering with varying fidelity have a significant impact on the computation cost both concerning latency and energy for batteryless sensors. For instance, in table 1, we show the latency and energy footprints of two typical operations present in inference pipelines of camera and microphone based data. These operational stages, i.e., data acquisition (e.g., sampling rate) and data pre-processing (e.g., reshaping, selecting different feature resolutions) offer unique optimisation opportunities if applied carefully.

Based on these observations, in the next section, we present ePerceptive, a framework for batteryless sensors offering energy reactive computation with a single DNN model.

3 ePerceptive PIPELINE OVERVIEW

The objective of ePerceptive is to adapt the computational complexity of DNN models to fluctuating energy conditions. ePerceptive focuses on vision and audio recognition workloads as we envisage batteryless sensors that can complete such workloads accurately and efficiently have the potential to produce high-value insights about the environment they are deployed in, at very low cost of deployment and maintenance. At the core of the framework we place an adaptive pipeline described in Figure 3. Since the overhead of a DNN inference is typically higher than the other stages, the objective is to adapt the pre-processing stages to reduce the execution overhead of the DNN with marginal loss in accuracy. The pipeline is designed to allow for adaptations at three different stages: data acquisition, featurisation and best-effort inference.

3.1 Data Acquisition Stage

During data acquisition the pipeline can choose to sample the sensors at different resolutions. In practice, this translates to capturing an image at higher or lower resolution for camera sensors (e.g., QQVGA or VGA) or changing the sampling rate of microphones (from 16kHz to 8kHz, for example). Higher sampling resolutions result in higher energy consumption for both cameras and microphones, hence ePerceptive can use this as *the first control knob* to adapt the computational overhead of the pipeline.

3.2 Featurisation Stage

At the second stage, adaptation is applied when raw data is transformed into features which are then used as input for the DNN model. For vision recognition DNN models work directly on the image pixels. However, since cameras often capture images only at a fixed resolution, in this stage ePerceptive resizes the image to a

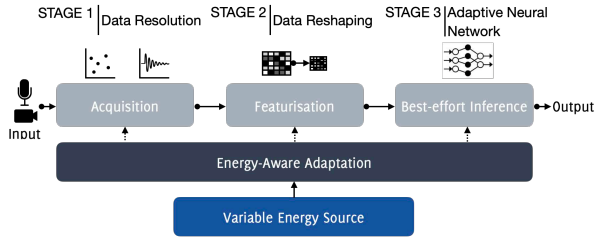


Fig. 3: ePerceptive high-level processing pipeline overview.

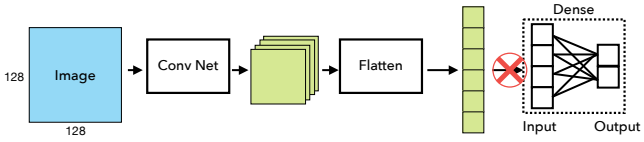


Fig. 4: Simplified architecture of a generic CNN built for a single resolution (e.g., 64×64). When presented with a larger input image, the Conv Net produces larger features which do not match with the input dimension of the dense layer. This model cannot compute.

lower resolution in order to speed up the DNN execution. For audio recognition, this step typically involves the computation of Mel Frequency Cepstral Coefficients (MFCCs). To obtain these features, the input audio signal of length L is divided into overlapping frames of length l, and frequency features are extracted for each frame. By adopting a stride parameter s, which determines the level of overlap, we can obtain "coarser" (if we increase it) or a "finer" (if we decease it) MFCCs. Intuitively, when the stride is small (i.e., bigger overlap between frames) many more frames will be considered and more features computed compared to when a larger stride is used. ePerceptive exploits this to adapt the MFCCs computation to variable energy availability. These dynamic resizing and variable MFCC computations represent *the second control knob* of the pipeline.

3.3 Best-Effort Inference Stage

All these adaptations produce input data at different quality levels (i.e., images at different resolutions or MFCCs computed from finer or coarser spectrograms). ePerceptive is capable of handling the various conditions with a single model and produces best-effort inferences reacting to the instantaneous available energy and to the degradation introduced by the previous stages.In the next sections, firstly, we introduce our multi-resolution inference approach to enable a single DNN model to produce accurate inferences even when the input data has a different resolution (§ 4). We then present a complementary multi-exit inference technique, that dynamically executes only a subset of the model layers albeit producing valid result (§ 5) to trade-off inference accuracy for latency. This represents *the third and last control knob* available to ePerceptive users to adapt inference computation to energy availability.

4 MULTI-RESOLUTION INFERENCE

In § 2, we have shown how energy/complexity adaptation of CNN models could be easily achieved by changing the input resolution. However, this becomes challenging on energy harvesting sensors due to the limited amount of memory available. Starting from these observations, the question we aim to answer in this section is: *can we design and train a model that can operate at multiple resolutions while maintaining a satisfying accuracy at each of them?*

4.1 Primer on DNN Input Pipelines

We first briefly review the characteristics of the input pipelines used to classify images and audio events. Vision models typically process one image at the time without any prior feature extraction stage. RGB images, characterised by three dimensions, height (H), width (W) and number of channels (C) are directly fed to the model's input stage, featurized using convolutional filters, and finally classified with one or more fully connected layers.

For audio models, commonly Mel Frequency Cepstral Coefficients (MFCCs) are extracted over a window of audio signal and fed to the DNN. When processing an audio signal of length L, the feature extraction consists in dividing the signal into overlapping frames of length l using a stride s to compute the spectrogram of the input. The number of frames considered is given by $T = \frac{L-l}{s} + 1$, and F features are computed for each frame, giving a total of $T \times F$ features. Hence, the audio input is still an image with H, W and C ($C = 1$) but the image represents frequency features over time. In this case, we can increase or reduce the input resolution by changing the stride used to compute the features.

4.2 The Problem with Multi-resolution

We analyse CNNs to understand why they cannot operate on different input resolutions. The main components of a CNN are 2D convolutional layers and dense (or fully connected) layers. When the network is created, the input dimension is assumed to be fixed and the weight matrices of all layers are initialised according to that input dimension and the other hyper-parameters of the model. This means that, to normally execute the network, the 3D features generated by the convolutional layers will be flattened to a vector whose length matches the input dimension of the following dense layer. If we feed the model a larger input (i.e., higher resolution image), the convolutional layers will easily produce larger features (with higher resolution). However, when these 3D features are flattened, just before the first dense layer, they will generate a longer vector which will not fit in the input of the dense layer, hence the network will not complete its computation. This is represented in Figure 4 where a model built for a smaller resolution is fed with a larger input resolution. The weight matrix of a dense layer is defined as $\mathbf{W} \in \mathbb{R}^{L \times M}$ where L and M are the lengths of the input and output vector, respectively. This implies that the only way for a dense layer to accommodate a larger (or smaller) input would be changing its weight matrix by re-training the model.

Contrary to dense layers, convolutional layers can produce features with different dimensions according to their input resolution, we show now how that is possible. The 2D convolutional layers of a DNN accept as input a 3D tensor ($\mathbf{I} \in \mathbb{R}^{H \times W \times C}$ with height H, width W and C channels) and produce a 3D tensor in output ($\mathbf{O} \in \mathbb{R}^{H' \times W' \times C'}$). The output is the result of convolving each kernel over the input tensor. The kernel tensor of a convolutional layer is defined by $\mathbf{W} \in \mathbb{R}^{S_1 \times S_2 \times C \times C'}$ while the bias vector by $\mathbf{b} \in \mathbb{R}^{C'}$, where S_1 and S_2 are the spatial dimensions of the filters, selected when the model is created. Similarly, C' (number of filters) is a

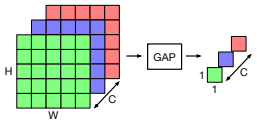


Fig. 5: Global Average Pooling (GAP) layer operation.

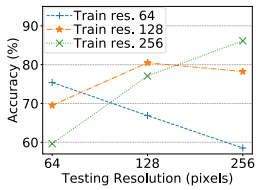


Fig. 6: MobileNetV1 models trained on a single resolution and tested on different resolutions. The dataset used for this experiment is Visual Wake Words [12].

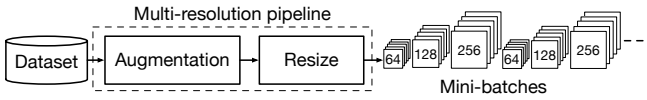


Fig. 7: Multi-resolution pipeline configured to train at 3 resolutions, 64, 128 and 256. For the audio pipeline, the *resize* component will change the stride used for the MFCCs computation producing features at different resolutions.

parameter of the layer itself and remains constant once the model has been built. The only dimension of **W** which depends on the layer's input is C, the number of input channels. Hence, as long as the input channels remain the same, the kernel tensor does not change, regardless of the input resolution. This allows convolutions to natively work on inputs at different resolutions without the need to modify the number of parameters. This is the property we exploit to adapt the model's computational complexity to instantaneous energy envelope.

4.3 Model Architecture

We have seen in the previous section that the convolutional layers support variable-sized inputs and generate variable-sized outputs. The objective, therefore, is to reshape the output of the last convolutional layer into a vector with fixed dimension, regardless of the input resolution, while maintaining enough information to classify the input accurately.

To achieve this goal, we borrow the concept of Global Average Pooling (GAP) [31] from established machine learning literature and apply it in our purpose-built model architecture designed to execute on batteryless sensors. GAP was introduced by Lin et al. [31] with the objective of replacing the last dense layer of a CNN model, removing many trainable parameters concentrated in the dense layers, with benefits in terms of reduced tendency to over-fitting, smaller models and speed of training. Given the advantages, the layer has been employed in several recent architectures, not as dense layer replacement but as dimensions reduction before the last dense layer(s) to limit over-fitting [23, 40]. In this work, we exploit the ability of a GAP layer in reducing the spatial dimensions of three-dimensional tensors: the average is computed across the entire spatial extent of the feature maps, producing a value for each channel. Therefore, a GAP layer reduces a tensor with dimensions $H \times W \times C$ to a tensor with size $1 \times 1 \times C$ as depicted in Figure 5.

By introducing a GAP layer after the last convolution we are able to shrink the spatial dimensions of the feature maps to a vector with a fixed size (e.g. $1 \times 1 \times C$), regardless of the input resolution. After the GAP layer one or more dense layers can be stacked because it is now guaranteed that their input dimension will not change. This simple but crucial modification enables a CNN to operate on inputs at different resolutions, benefiting also from the GAP's advantages mentioned earlier. In the specific context of embedded machine learning, where onboard memory is limited, introducing a GAP layer brings also important savings in terms of memory consumption as dense layers typically hold the largest number of parameters. However, we will demonstrate in § 7.2 that adding a GAP layer is not sufficient to ensure good accuracy at different resolutions, hence the introduction of our multi-resolution training strategy.

4.4 Model Training

The GAP layer ensures that the feature maps having different dimensions generated by the convolutional layers reduce to a vector with a fixed dimension. However, when the model is trained with a single input resolution, the kernels learn to extract meaningful features only at that resolution and the learned parameters do not translate well to other resolutions. To demonstrate this behaviour we trained three MobileNetV1 [23] models at resolutions 64×64, 128×128 and 256×256 on the Visual Wake Words dataset [12]. We then tested each model at the three resolutions. Figure 6 reports the accuracy of the models. We observe that, as expected, each model achieves the highest accuracy at the resolution they have been trained on. However, when the testing resolution is changed the accuracy drops significantly. We see a similar behaviour in the other datasets we consider in this work, including audio (see § 7.2).

To overcome this issue we devise a training strategy that takes inspiration from data augmentation techniques. When data augmentation is added to the training process, dataset samples are randomly transformed during training. When dealing with images for examples, typical augmentations include horizontal flip, rotation or colour alteration. This essentially expands the available dataset and helps the model generalise better. Following the same principle we propose to vary the input resolution during the training process and force the model to learn features that are resolution invariant. Since the model is presented with input where the features of the target object change in scale continuously, the kernels, instead of specialising on a single resolution are forced to adapt to the different scales and spatial relationships between the features.

To implement this strategy, we design a multi-resolution input pipeline which, during training, resizes the input for each mini-batch. Individual mini-batches cannot contain input with different resolutions but it is possible to re-scale between each mini-batch. Before input resizing we apply classic data augmentation techniques (e.g., horizontal flip and rotation for images and noise addition for audio) to expand the dataset. Figure 7 shows the pipeline operation. At training time, our pipeline starts from the first resolution, loads the first mini-batch, applies classic data augmentation, resizes all

the samples to the selected resolution and passes it through the model. Then the pipeline selects the next resolution and creates the next mini-batch. Once all the resolutions have been used the process starts over from the first one. At the end of each epoch we shuffle the entire dataset to ensure that the same resolutions are not used on the same images. This pipeline guarantees that all resolutions are used during training. The audio pipeline is the same with the only difference that the resize operation is performed by changing the stride used for MFCCs computation.

5 MULTI-EXIT INFERENCE

The multi-resolution architecture and training strategy enables a single model to exploit different input resolutions to trade-off accuracy for latency. Another approach would be to use models that have a different number of layers. In energy harvesting devices, only few models can be stored, therefore, it is difficult to achieve the needed adaptation to match a fluctuating energy envelope. Hence, the driving question of this section is: *can we design and train a model that can adapt its depth dynamically at inference time?*

5.1 Model Architecture

To achieve the needed flexibility, we take inspiration from a class of algorithms called *anytime* [7, 41]. The central premise behind them is the ability to be interrupted before workload completion—say as a result of energy depletion—while retaining the ability to output a valid, albeit degraded, result. Further, a wider energy envelope would allow to progressively refine the earlier approximate result. Anytime algorithms have been originally used for time-dependent planning and decision-making [15]. Such paradigm aligns well with the unpredictable nature of energy harvesting systems as anytime algorithms can approximate results to available power budget, thereby guaranteeing a timely response. That is, an anytime algorithm is able to trade off confidence in results for energy and compute efficiency.

Grounded on these observations, we borrow this *anytime* paradigm into ePerceptive contextualising it in batteryless sensor settings. Accordingly, we enhance the model architecture by introducing exit points at intermediate positions throughout the network. A classifier, typically implemented with one or more dense layers, is placed at the end of each exit point to produce auxiliary outputs in addition to the classic output of the network at the last layer. This approach effectively reduces the number of layers that the input needs to traverse, hence achieving reduction in latency at the expense of lower accuracy. The principle behind this method is that DNNs extract better features as the model gets deeper. To best of our knowledge, this is the first work that brings the anytime paradigm in a battery-less context.

The proposed architecture brings several advantages for battery-less sensors as it allows at runtime to select the appropriate exit based on the instantaneous energy budget. Additionally, this decision can be updated in case the amount of energy being harvested changes. For example, considering a model with three exits, at some point in time the exit #2 could be selected. If, during execution, the harvested energy is reduced (e.g., passing cloud) the system can select the exit #1 (assuming it did not pass that exit point already)

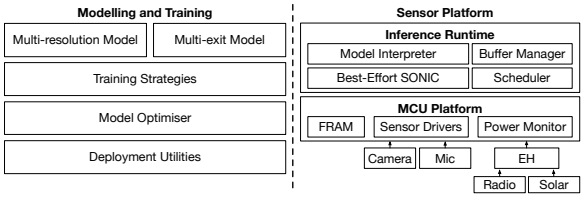


Fig. 8: Architecture of ePerceptive. Left are the components used offline for model training and deployment, right are the components running on the MCU platform.

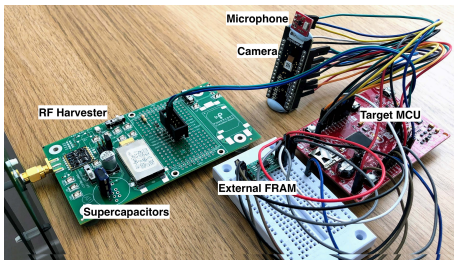


Fig. 9: Hardware setup.

and produce a degraded inference accuracy-wise with shorter latency. Similarly, if the harvested energy increases, a later exit point could be selected to increase the inference confidence.

5.2 Model Training

In order to have valid outputs in all exit points, the loss function needs to have a component for each exit and the training objective is to optimise this loss for all exits. This way each output can propagate its prediction error in relation to the ground truth and train the layers that precede the exit point. We denote the cross-entropy loss function for each exit with $L_n(\hat{y}_n, y; \theta)$, where $\hat{y}_n$ is the prediction produced by exit n, y is the ground truth and θ are the weights of the layers between the model's input and the exit n. To train the entire network at once we optimise the following weighted loss function:

$$L_{model}(\hat{y}, y; \theta) = \sum_{n=1}^{N} w_n L_n(\hat{y}_k, y; \theta). \quad (1)$$

N is the number of exits and $w_n \in [0, 1]$ are the weights applied to the loss function of each exit point. The weights allow to balance the importance of each exit during gradient descent and can be used to control the accuracy on each of them. Increasing the weight at a specific exit point forces the network to learn better features at the preceding layers, hence, potentially producing better results.

During training, the predictions of all exit points are computed, compared with the ground truth via the loss function and then the weights are updated. At inference time instead, we modify the model to select the exit point for each input sample. In § 7, we show how this enables to scale the inference complexity based on the available energy by selecting the appropriate exit at runtime.

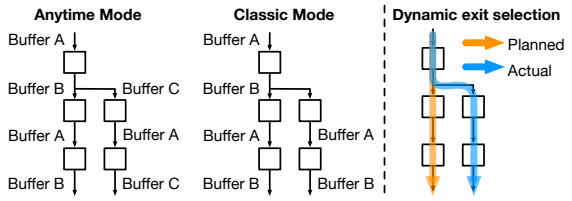


Fig. 10: Interpreter buffer management and dynamic exit selection.

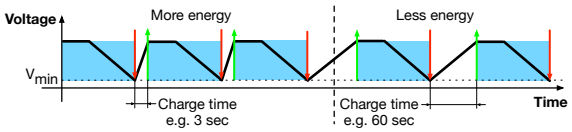


Fig. 11: Simplified supercapacitor voltage fluctuations when there are different levels of energy available. Blue periods represents intervals when the device is running. Red arrows represent when the device enters LPM3 and green arrows when the device wakes up.

6 ePerceptive IMPLEMENTATION

6.1 Hardware

ePerceptive implementation consisted of three main hardware units: MCU, sensors, and harvester. We used TI's MSP430FR5994 MCU which has 256KB of embedded FRAM, 8KB of SRAM and 16 MHz CPU speed. For vision application, we interfaced the MCU with the Himax HM01B0 ultra low power CMOS Image Sensor, consuming <2mW at QVGA, 30FPS. The Analog Devices's ADMP401, a low power omnidirectional MEMS microphone was used as the sensor for the audio application. We used two kinds of harvesters; an RF Harvester (Powercast TX91501B, 915 MHz, 3W transmitter) was combined with the Powercast P2110B controller and a solar panel (IXYS SLMD600H10L) was controlled with the TI's BQ25570 power management chip.

6.2 Software

The software implementation of ePerceptive consists of two distinct components, an offline framework for training, optimising and provisioning the model to batteryless sensors, and an on-device framework for runtime inference guided by a lightweight scheduler. These components are depicted in Figure 8.

6.2.1 Offline Framework: This component consists of a package of valuable tools to seamlessly guide the definition of Tensorflow adaptive models, training strategies described above for multi-resolution and multi-exit and optimisations for execution on constrained devices. We provide a range of optimisation techniques like singular value decomposition (SVD), Tucker tensor decomposition, pruning and batch normalisation (BN) folding. The optimiser automatically computes the runtime memory requirements and applies the various optimisations to different parts of the model in an iterative manner until it fits on the device while making sure there is little to no drop in accuracy. Additionally, this layer supplies the tools to export and flash the models into the sensor platform.

6.2.2 On-Device Framework: In the original SONIC implementation, since the model is pre-determined at compile time and does not change at runtime, its structure and the management of the intermediate buffers are hard-coded in the firmware. This is not suitable for ePerceptive, because at runtime the sequence of layers executed might change from inference to inference when different exits are selected and the intermediate buffers that hold the layers' activations need to be expanded or shrunk based on the input resolution. Besides, we also need a mechanism to dynamically select the best execution path in response to available energy. To this end, we extended the SONIC framework with three distinct modules.

Model Interpreter: We designed and implemented a model interpreter that reads the definition of the model from FRAM and executes it in an intermittent-safe way. The definition of the model is a table that contains the type of each layer (e.g., convolution or dense layer), their configuration including bias, padding and stride and a pointer to the memory location where the weights are stored. Crucially, it also stores the possible resolutions supported by each layer and the locations within the model where early-exits can be taken. This definition is automatically generated by the offline tools at the the end of the model training. At runtime, the interpreter reads the definition of each layer and executes it using the SONIC primitives.

Buffer Manager: The buffer manager (BM) manages and re-uses the input/output buffers stored in FRAM for each layer as depicted in Figure 10. The BM is used by the interpreter during the execution of the model. Two modalities are supported. In the *Anytime Mode* the buffer manager enables the anytime paradigm which allows an input sample to be processed by further layers after an initial result has been produced in an earlier exit. In this case, the BM allocates an additional buffer C which is used to complete the execution of the early exit while B remains untouched and can be used to continue to a deeper exit later. The *Classic Mode* instead requires only two buffers but does not allow to continue the execution after an early-exit has been taken.

In addition, this component keeps track of the execution of the model using the FRAM to ensure that the data structures are consistent even in case of power failures. Additionally, as shown in Figure 10, the buffer manager allows to dynamically change the exit to take based on the available energy even if this differs from the one originally planned by the scheduler.

Scheduler: ePerceptive processes input samples every time there is sufficient energy to provide inference outputs as promptly as possible. This light-weight scheduler decides which input resolution and early exit to take for the next inference by estimating the current energy available in the environment. The scheduler achieves this by exploiting a key principle - the charging rate of the supercapacitor is proportional to the available environmental energy. This implies, when the environmental energy is high (e.g., during a sunny day), the downtime of a batteryless sensor will be shorter as the capacitor will charge faster and vice versa when the energy is scarce. Figure 11 exemplifies this. We estimate the available energy by measuring the time taken by the capacitor to charge up to a sufficient voltage. As shown in Figure 11, we monitor the voltage across the supercapacitor with the internal comparator and enter in Low Power Mode 3 (LPM3) when the voltage drops below V_{min}. While in LPM3 we use the RTC (Real Time Clock) to

measure the time taken by the capacitor to charge until the device wakes up.

We use solar and RF traces to determine the distribution of charge times. Based on this distribution, we build a lookup table that associates a configuration (represented as a tuple <resolution, exit>) to a range of charge times. The table is built to balance latency and accuracy by selecting configurations with lower latency (i.e., lower accuracy) when energy is scarce and configurations with higher latency (i.e., higher accuracy) when more energy is available. At runtime, when the device wakes up (represented by green arrows in Figure 11), after reading the charge time from the RTC, we select the configuration that corresponds to the current charge time. The table lookup has a negligible cost given that the table has as many entries as the number of configurations (i.e., at most 7 in our experiments). In case the device loses power completely, for example during the night for a solar-powered system, at the next wake up, since the system cannot estimate the amount of energy because the RTC was off, it selects the configuration with the lowest latency to provide a prompt inference.

Our practical approach gives an indication of the current amount of energy available and it is used by the system to take a *short term* decision on the next configuration to run. Please note that one can consider a sophisticated scheduler with long term energy forecasting building on the rich literature on real-time task scheduling and energy prediction. However, we consider such a mechanism is a future avenue of our work. In the current context, however, we want to highlight that ePerceptive allows and support the implementation of different policies.

7 EVALUATION

7.1 Methodology

Datasets: To evaluate the generalisation capability of our framework, we chose three datasets that reflect tasks that could potentially be achieved by low-power devices in the wild, where energy is scarce.

Visual Wake Words [12] is a benchmark dataset for tiny vision models purposefully built to execute on microcontrollers. It contains images belonging to two classes, whether a person is present in the image or not. This dataset offers high-resolution images and the training and validation sets include 115k samples while the test set comprises a total of 8k images.

Camera CATalogue [44] dataset consists of animal images captured from 750 cameras in South Africa. These snapshots contain 55 animal species for a total of 520K images. In this specific work, we focus on classifying the presence or not of an animal in an image discarding the info about the species.

Speech Commands v0.02 [43] dataset is a collection of 105k 1-second long utterances of 35 words meant for keyword detection. In our setting, we use ten keyword classes *yes, no, up, down, left, right, on, off, stop, go* and the rest falls in the "Unknown" class. The input to the model is a two-dimensional tensor extracted from the one-second-long audio recording, consisting of time frames on one axis and 12 MFCC features on the other axis.

Models: ePerceptive adapts the computational complexity of two realistic off-the-shelf neural networks, enabling them to react to fluctuating energy conditions typical of battery-less sensors. This adaptation happens at both data processing and inference stages.

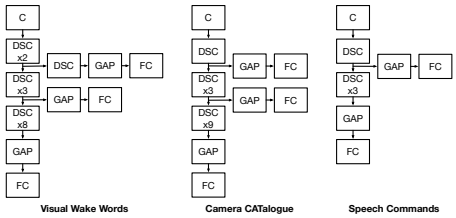


Fig. 12: Early-exit architectures.

For our vision tasks, we use MobileNetV1 [23] - a state-of-the-art CNN architecture particularly suitable for low-power settings. It consists of 14 blocks, each composed of 2D depthwise convolution layers and 1x1 (point-wise) convolutions.

For our audio task, we utilise a depthwise separable CNN architecture, proposed in [48]. This architecture is implemented on a smaller version of MobileNetV1 and is composed of 1 regular convolution layer, four DSCs and a fully connected layer.

As mentioned before, the original models are enhanced to support multi-resolution input and multi-exit branches. To achieve this, we perform a thorough hyperparameter search and find the best-performing ones for the vision and audio tasks. The final model architectures are depicted in Figure 12. Global Average Pooling layers are inserted before the fully connected layer of each exit point to support multi-resolution inputs at all exits.

Performance metrics: We consider different metrics for the evaluation: accuracy, latency, energy, and number of samples processed in a fraction of time.

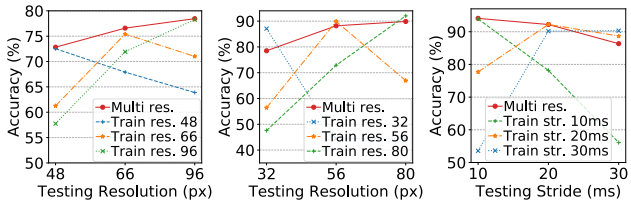
7.2 Constant Power Microbenchmarks

In this section we evaluate ePerceptive when running with constant power. These benchmarks are meant to assess how well the two techniques enable a graceful trade-off between inference accuracy and energy. In the following experiments we profile only the inference stage of the pipeline.

Multi-resolution models. Figure 13 shows the accuracy of our multi-resolution models when tested at different resolutions (x-axis) compared to the accuracy of models trained at a single resolution. The multi-resolution models achieve high accuracy at the three resolutions while the other models show high performance only at the single resolution they have been trained on, with significant drop for the others. This result is consistent for the three datasets. Notice that for Speech Commands the accuracy decreases as the stride increases because larger strides produces smaller and coarser MFCC features making the classification task more difficult.

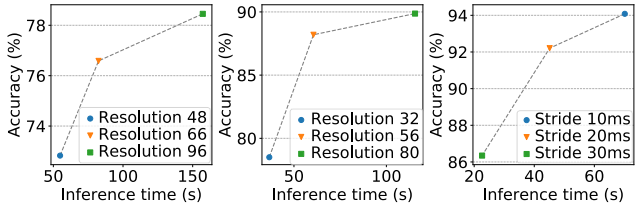
Interestingly, in some cases, the accuracy of the multi-resolution model is slightly higher than the other models at the same resolutions. For the Visual Wake Words dataset this is quite marginal for resolutions 48 and 96 for which the multi-scale model is 0.3% and 0.2% more accurate, respectively. However it is more significant for resolution 66 for which the accuracy gain of the multi-scale model is 1.2% compared to model only trained at that resolution.

For the other models, the drop in accuracy with respect to the best model at a particular resolution is within 4%. The only exception is the Camera CATalogue model which experiences an 8% drop at



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 13: Accuracy of multi-resolution models compared to models trained on individual resolutions.



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 14: Accuracy and latency of multi-resolution models when running with constant power.

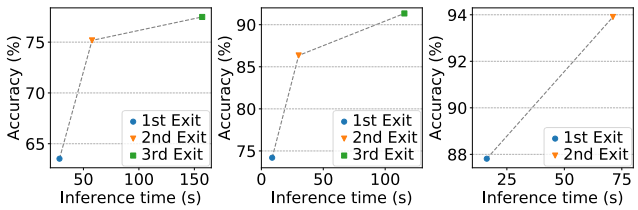
resolution 32. We hypothesise this is because 32×32 is a small resolution for image classification and the higher accuracy drop might be an indication that this is the limit at which the model is capable of producing reasonable inferences (i.e., with accuracy above 70%).

Figure 14 shows how the change in input resolution translates directly in latency gains. The latency is reduced significantly for all models while the accuracy difference between higher and lower resolution is around 11% for the Camera CATalogue model, or lower. For the audio model the benefit of operating at multiple input resolutions is also capitalised at the feature extraction stage. In fact, computing MFCCs with a larger stride reduces the complexity of the features extractor reducing its latency. Table 1 shows that for the frame length an stride parameters we have chosen there is a 3x latency and energy reduction.

These results shows that our multi-resolution model and training strategy enable a single model to operate accurately at different resolutions without the overhead of storing the parameters of different models in memory.

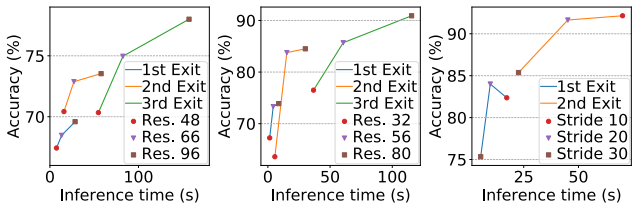
Multi-exit models. We now move to the evaluation of the second technique that is part of ePerceptive: multi-exit DNNs. The latency results are reported in Figure 15. They show how this is another effective knob to control the complexity of DNNs inferences without heavy overhead in terms of memory consumption. In fact these models are only slightly larger that the non-multi-exit counterpart because they have a limited amount of layers in the exit branches. Contrary to the multi-resolution approach described earlier we see a more significant accuracy variation over the various exits. This is due to the fact that in this case the model is severely reduced when an early exit is taken. For example the first exits skips the majority of the network.

Combination of multi-resolution and multi-exit models. An interesting advantage of the techniques we just described is that they can be combined in a single model. Figure 16 shows the results for the three datasets when models have been trained combining



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 15: Accuracy and latency of multi-exit models when running on constant power.



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 16: Accuracy and latency of models combining multi-resolution and multi-exit techniques when running with constant power.

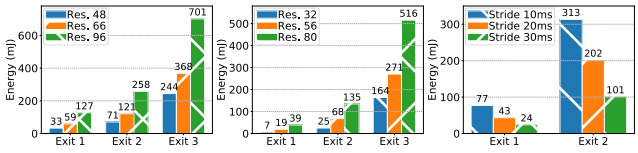
multi-exits and multi-resolutions. We observe how the combination of exits and resolutions create a smooth trade-off between accuracy and compute. At any given point, at runtime, the system can alter the model complexity by picking one of the *<resolution, exit>* combinations. Combinations with higher latency typically result in higher accuracy and vice-versa. However, not all configurations are useful because in some cases there could be overlap between them. An example of this is resolution 96 on the first exit for Visual Wake Words. That configuration is not particularly useful because there is an additional one (resolution 48, exit 2) that achieves higher accuracy with slightly lower latency (other cases are present in the other datasets too). This is the result of the interaction between the two techniques. Even if a higher resolution typically corresponds to higher accuracy, the introduction of an exit point with higher computational capacity could in a way revert that by achieving higher accuracy at a lower resolution.

The smooth latency transition translates directly into energy consumption adaptation. Figure 17 shows how each configuration has a specific energy profile. These results demonstrate how ePerceptive enables the adaptation of DNNs' complexity to achieve gains in terms of latency and energy consumption.

Memory Usage. Table 2 reports the amount of memory required to store the weights of different MobileNetV1 versions. The first two rows represent models that are equivalent to our multi-exit model but have only one, or two exits. The third row instead is the memory required by the classic MobileNetV1 model at a single resolution. We observe that our multi-exit, multi-resolution model requires slightly more memory than the classic model since it needs to store the weights of the layers in the additional exits. When using individual models, in order to have different latency-accuracy operating points at runtime, the system should store weights for each of them. For example, with our Visual Wake Words model we can have 9 different operating points (3 resolutions and 3 early exits) using 171KB for the weights. The same 9 operating points using individual models

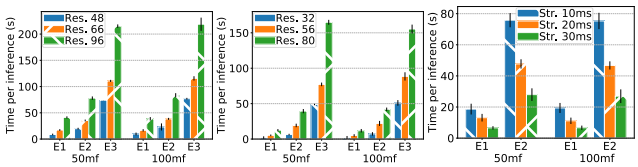
Table 2: Memory required for storing weights of different models.

Model	Weights Memory (KB)
MobileNetV1 First Exit Only	2.2
MobileNetV1 Second Exit Only	12.6
MobileNetV1 Full Model	169.6
ePerceptive MobileNetV1	171.0



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 17: Average energy per inference of models combining multi-resolution and multi-exit techniques with constant power.



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 18: Average time per inference of models combining multi-resolution and multi-exit techniques when running with RF intermittent power. Two supercapacitors are used, 50mF and 100mF.

will require $2.2 \times 3 + 12.6 \times 3 + 169.6 \times 3 = 553.2KB$. This is more than three times more memory than when using a single model that can support multi-resolution inputs and early-exits. Although, techniques exist to reduce the memory footprint of DNNs, they are applicable also to our single model and the overhead of having to store multiple models rather the only one remains.

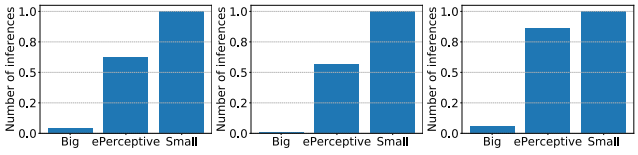
7.3 Intermittent Power Microbenchmarks

In this section we demonstrate that ePerceptive operates robustly when power is intermittent and computation cannot be sustained. This is crucial for the applications we are considering in this work where inexpensive sensors can be deployed at scale with limited need for maintenance to recharge or replace batteries.

ePerceptive builds on SONIC [18], therefore an intermittent-safe implementation of DNNs primitives was already available. However, with our extension of SONIC to support multi-resolution and multi-exit models, it is imperative to maintain and update intermediate decisions (e.g., early exit taken or shape of intermediate feature maps) across power failures. To asses the correct operation of such elements, we position the RF Powercast transmitter 1.5 meters away from the receiver and use the receiver to power the MSP430 running the combined multi-resolution, multi-exit models. We use a 100mF and 50mF supercapacitor for the experiments. From Figure 18 we observe that the overall latency of the various configurations increases, due to the periods when the supercapacitor recharges. Despite that, the trends we observe are the same we saw when the device was continuously powered.

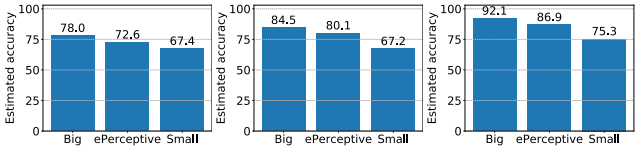
7.4 End-to-end System

In this section we evaluate the end-to-end performance of ePerceptive using solar and radio energy harvesting.



(a) Visual Wake Words (b) Camera CATalogue. (c) Speech Commands.

Fig. 19: Number of normalised inferences performed by the three models when running from RF and solar harvesting traces.



(a) Visual Wake Words. (b) Camera CATalogue. (c) Speech Commands.

Fig. 20: Estimated accuracy of the three models when running from RF and solar harvesting traces.

Energy traces: In order to obtain more reproducible results, we recorded harvesting traces using the Ekho tool [20] and then run the system on emulated traces. For RF harvesting, we simulated an indoor deployment and placed the RF receiver and transmitter in front of each other at a minimum distance of 70cm. The receiver was moved every few minutes to various distances (up to 3.5 meters), and its antenna orientation was changed to simulate harvesting attenuation caused by people moving in the environment and obstructing the path between transmitter and receiver. We recorded a 30-minute long trace. For solar harvesting, we simulated an outdoor environment by building a controllable light intensity setup using dimmable light bulbs. Similar to the RF trace, we varied the light intensity every few minutes. Since our system would run without interruption with 5klux or more, we kept the intensity below this threshold to ensure the system would run intermittently. Also, in this case, we collected a 30-minute long trance. For all the evaluations, the system was equipped with a 50mF supercapacitor.

Configurations: We apply the Visual Wake Words and Speech Commands workloads on the RF trace since they represent typical indoor use cases of low power sensors (i.e., people and sound detection). We apply the Camera CATalogue workload on the solar harvesting trace, which is more representative of outdoor sensing applications. We use the ePerceptive models presented in the previous sections that support three input resolutions and three exits for the vision models and two exits for the audio model.

Baselines and metrics: As baselines, we use two static models that cannot adapt their computation to instant energy availability. We selected these models to have particular characteristics. The first model is efficient, fast but does not provide the best accuracy (we call this model *small*). This model is representative of an application that prioritises low latency over accuracy. The second model (*big*) is at the other end of the spectrum. This model is more accurate but slow at executing; hence the priority is on accuracy. Ideally, these two models form a lower and upper bound on the performance of ePerceptive which instead tries to obtain a more balanced performance.

For the evaluation metrics, we use the total number of inferences computed by the models and the estimated accuracy obtained by the models. We normalise the total inferences by the number of inferences of the small model to have a more precise comparison. This is because the small model is the fastest one, and it will always produce a higher number of inferences.

Given the difficulty of acquiring reliable ground truth while running the system intermittently or for specific tasks (i.e., animal spotting), we estimate the accuracy that the models would achieve by using their performance on the test set. For the two static models, this corresponds directly to the test set accuracy. For the ePerceptive models, we compute a weighted accuracy based on the number of inferences performed for each resolution/exit combination and their accuracy on the test set.

Results: Figures 19 and 20 report the results of these experiments. We observe how ePerceptive is capable of obtaining a balanced performance by completing more inferences than the big static model, however without compromising significantly the overall accuracy. The Visual Wake Words and Camera CATalogue models have similar inference times (recall Figure 16), so the number of inferences they can complete is similar. The Speech commands model is the one with shorter inferences overall; hence it manages to complete more. From Figure 20 we notice how the adaptation performed by ePerceptive ensures accuracy improvements compared to the small model up to 13% for the Camera CATalogue dataset and around 5% and 11% for the Visual Wake Words and Speech Commands datasets, respectively. Compared to the big model instead, ePerceptive shows a modest drop in accuracy around 5% across the three datasets. Collectively, these results show how ePerceptive offers a more balanced performance across the two important metrics of inference throughput and accuracy.

8 DISCUSSION

The Dynamics of Flexibility and Overhead: ePerceptive aims at improving inference throughput on batteryless sensors by dynamically scaling the model's complexity in response to fluctuating energy availability. For this reason we adopted an energy-oriented approach to decide which resolution and exit to use for the model computation as opposed to an approach based on the expected accuracy. In our current implementation, such decision is taken on a short term basis by estimating the current energy level. This benefits from a negligible scheduling overhead, since the selection mechanism involves only a table lookup guided by the RTC, that measures the capacitor charge time running in LPM3 drawing around 700nA. By contrast, previous work, especially in the context of early-exit networks, use additional resources to classify the input multiple times and stop when a certain accuracy is reached. However, this is counterproductive for energy harvesting sensors since it will incur additional power failures reducing throughput.

The Dynamics of Accuracy and Latency: We envision the utility of ePerceptive -powered sensors will be maximised in non-mission critical applications where large scale and temporal coverage at minimum maintenance cost is paramount. For example, in an audio-visual monitoring system for outdoor industrial settings or wildlife conservation applications. The benefit of batteryless systems is to enable large deployments of hundreds of devices in such settings without the need to recharge or replace batteries. Accordingly, we argue it is imperative to balance latency and accuracy to ensure a more efficient usage of the limited energy and obtain an increased number of predictions. By adapting the model's latency, while accepting a slightly degraded accuracy performance, we make a better usage of energy and can achieve an increased throughput with ePerceptive. Regardless, the two techniques we have presented are amenable to be used in conjunction with different latency-accuracy trade off policies. Applications that require an higher level of accuracy for each inference could select larger input resolutions and deeper exits to ensure a superior recognition performance. An example of such applications is visual intrusion detection where the cost of wrong classifications is higher than in monitoring applications, such as in wildlife conservation. Nevertheless, we consider our techniques are useful to adapt model complexity at runtime and are independent from the specific policy.

The Dynamics of Sampling Rate and Throughput: With ePerceptive, every time there is energy to sustain computation the device processes an input. This design ensures capturing as many interesting events as possible. Adopting a different strategy where the sampling rate of the system is reduced would mean that the device would not sample its sensors and run inferences on them even if there is sufficient energy. This would result in saving energy for a later time but completing fewer inferences overall. Waiting to sample at predefined intervals might lead to missing interesting events that could instead be detected. However, we also acknowledge that, in the presence of external triggers, for instance with motion, preemptive sampling with a lower rate would benefit from ePerceptive and would definitely improve overall utility. We plan to add such functionalities to ePerceptive in our future work.

The Dynamics of Resolutions and Exits: Through our experiments and evaluation, we have identified several guidelines for designing ePerceptive models that ensure a good balance between latency and accuracy. For multi-resolution DNNs, our empirical observations highlight that to achieve an appreciable increase in accuracy, a considerable gap between the resolutions/strides is needed. This design allows for a neater separation between the different scales and enables the convolution layers to capture a substantial amount of extra information to incorporate into their representational capability. Similarly, for the exit branches, sufficient distance between two consecutive branches allows the layers in between to extract meaningful features which increases the accuracy at the next exit. If the exits are too close to each other, the accuracy difference between the two would be minimal. This could be counteracted by adding additional layers in the early exit branch, as we did for the Visual Wake Words model. The important thing to consider in this case is that the computational complexity of that specific early exit should not overcome the complexity of the next exit. These considerations are crucial for a well-balanced multi-resolution and multi-exit ePerceptive model for batteryless sensors.

9 RELATED WORK

9.1 Intermittent Computing

Devices running on harvested energy were initially envisioned with energy-neutral objectives [27], however, more recently, the

design objective has shifted to intermittent computing, an execution style that spans multiple power failures [33]. Power failures erase volatile memory and registers needing frequent checkpoints, where volatile state is stored in non-volatile memory. Energy fluctuates significantly and is bursty in nature. The net effect is an uptime of few milliseconds only and with periods of power denial that could span hours. In such peculiar environment, ensuring the forward progression of computational workloads despite power failures, as tackled by several previous works [3, 22, 34, 47], is crucial.

With specific focus on DNN execution on intermittent platforms, Gobieski et al. [18] argued that inference accuracy dominates performance improvements and it is essential to reduce loss of energy from unnecessary communication of raw signals. This has motivated to improve inference on harvested energy platforms by using DNNs and the development of SONIC [18], which implements intermittent DNN primitives on TI MSP430 platforms.

ePerceptive builds upon SONIC and previous work in the intermittent computing community. ePerceptive uses the non-volatile memory checkpoint systems but goes beyond them by enabling DNN models to dynamically scale their latency and energy requirements at runtime. With the integration of two complementary techniques into SONIC and the implementation of a flexible model interpreter we show how we are able to run realistic CNN models on extremely constrained devices powered by energy harvesting achieving good performance on challenging datasets.

9.2 Efficient and Adaptive DNNs

Recently, traditional DNNs are modified to fit in memory, increase execution speed, and decrease energy demand to enable efficient inference on edge devices. Techniques like pruning redundant weights and filters during or after training and later fine-tuning the model helps in decreasing model size without high loss in accuracy [9, 11, 19, 45, 46]. They can produce a family of networks with different computational requirements however require significant offline processing to produce and fine-tune these networks [9, 11, 45, 46]. These algorithms aim at reducing the complexity of a single model in order to run it on constrained devices but do not consider the problem of adapting the model execution to match fluctuating energy. In fact, to obtain dynamic resource scaling at runtime, the networks produced by these frameworks would need to be stored on the target device with significant memory overhead. This is prohibitive for the platforms we target in this work. Instead, we propose to use a single model, that can be trained in a single session and that offers variable performance at runtime.

Tan et al. [40] implemented DNN scaling by changing the number of layers, number of filters and feature maps resolution to improve accuracy. Though, they train different models with varying complexity rather than applying all techniques to a single model raising many questions about their memory footprint. Similarly, in Chameleon, different knobs like frame rate, resolution, type of inference model is tuned based on temporal and spatial information for video processing to minimise computation [26]. In terms of dynamic resource adaptation at runtime, Fang et al. presents a multi-capacity model that can dynamically adapt its resource utilisation at the expense of accuracy degradation (NestDNN [17]). The model is obtained by first pruning filters from an existing network and then iteratively re-introducing them. At runtime, the filters to use can be selected based on the available resources and desired accuracy. In ePerceptive we propose two different techniques applicable to vision and audio models. Additionally, NestDNN requires a more expensive and more complicated training procedure since the model has to be re-trained several times: during the initial pruning to obtain the seed model and during the filter growing stage to obtain the multi-capacity model. Our approach is instead simpler since it requires only one training session but yet not less effective given that we can obtain several accuracy-latency operating points at runtime, similar to what NestDNN achieves.

The early-exit functionality of ePerceptive is inspired by a class of multi-exit anytime algorithms [8, 24, 30, 36, 37, 41, 42] whose aim is improving inference time by predicting easy inputs at early stages. However, the proposed concepts do not consider memory-constrained embedded devices and intermittent energy scenarios.

In ePerceptive we adopted an energy-oriented approach to decide which resolution to use and what model's exit to take. The decisions are based on the short term estimated available energy. Previous works adopt a different policy and decide to exit the network when a target accuracy is reached [8, 24, 41]. In [24] and [41], the decision to exit the network is taken based on the output confidence of each branch, meaning that the branches need to be executed until the end to determine if the input should exit or not. For inputs that need to exit at deeper branches, this constitutes overhead and wasted computation/energy. Similarly, in [8] the network contains decision functions at the branching points to decide if an example should go out or continue further down the network. In practice, these functions are implemented as dense layers and trained iteratively one after the other. Our first choice for ePerceptive is to avoid these approaches to make sure that the majority of the energy we spend goes into the network execution progress and not into decision policies. Our target devices are extremely resource constrained and cannot reach state-of-the-art results in terms of model accuracy. Hence, focusing excessively on model accuracy might be counterproductive. Trying to achieve higher accuracy on intermittent systems would result in much longer inference times because the device would have to store and load its state several times, limiting the model's throughput.

10 CONCLUSION

In this paper, we present the design and implementation of ePerceptive: a novel framework for best-effort embedded inference on batteryless sensors. At the core of this framework, there are two complementary mechanisms devised specially to allow dynamic adaptation of existing DNN models, without incurring memory overhead. First, ePerceptive enables training a single DNN model which performs accurately at multiple input resolutions without the overhead of storing several models in memory. Second, ePerceptive modifies the DNN architecture to provide valid inferences even if interrupted before completion. The combination of these techniques offers a controlled grading of performance, and is used to match the computational load to the available power budget. With ePerceptive, we have designed batteryless cameras and microphones and evaluated them with multiple vision and acoustic workloads that demonstrates the efficacy of our principled approach in maximising the overall system availability.

REFERENCES

[1] Mattia Antonini, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. 2019. Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things (AIChallengeIoT'19)*. 7. https://doi.org/10.1145/3363347.3363363

[2] Omid Ardakanian, Arka Bhattacharya, and David Culler. 2016. Non-intrusive techniques for establishing occupancy related energy savings in commercial buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*. 21–30.

[3] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. 2014. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters* 7, 1 (2014), 15–18.

[4] Sara Beery, Grant Van Horn, Oisin MacAodha, and Pietro Perona. 2019. The iwildcam 2018 challenge dataset. *arXiv preprint arXiv:1904.05986* (2019).

[5] Sara Beery, Grant Van Horn, and Pietro Perona. 2018. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[6] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*.

[7] Mark Boddy and Thomas L Dean. 1989. *Solving time-dependent planning problems*. Brown University, Department of Computer Science.

[8] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for fast test-time prediction. *arXiv preprint arXiv:1702.07811* (2017).

[9] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[10] Gregory Chen, Hassan Ghaed, Razi-ul Haque, Michael Wieckowski, Yejoong Kim, Gyouho Kim, David Fick, Daeyeon Kim, Mingoo Seok, Kensall Wise, et al. 2011. A cubic-millimeter energy-autonomous wireless intraocular pressure monitor. In *2011 IEEE International Solid-State Circuits Conference*. IEEE, 310–312.

[11] Y Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2018. Understanding the limitations of existing energy-efficient design approaches for deep neural networks. *Energy* 2, L1 (2018), L3.

[12] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. *arXiv preprint arXiv:1906.05721* (2019).

[13] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 767–781.

[14] Jasper de Winkel, Carlo Delle Donne, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. 2020. Reliable Timekeeping for Intermittent Computing. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 53–67.

[15] Thomas L Dean and Mark S Boddy. 1988. An Analysis of Time-Dependent Planning.. In *AAAI*, Vol. 88. 49–54.

[16] Andy Rosales Elias, Nevena Golubovic, Chandra Krintz, and Rich Wolski. 2017. Where's the bear? Automating wildlife image processing using iot and edge cloud systems. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 247–258.

[17] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 115–127.

[18] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 199–213.

[19] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[20] Josiah Hester, Timothy Scott, and Jacob Sorber. 2014. Ekho: Realistic and repeatable experimentation for tiny energy-harvesting sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*. 330–331.

[21] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[22] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[24] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844* (2017).

[25] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[26] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*.

[27] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B Srivastava. 2007. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)* 6, 4 (2007), 32–es.

[28] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530* (2015).

[29] Seulki Lee and Shahriar Nirjon. 2019. Neuro. ZERO: a zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 138–152.

[30] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[31] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[32] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[33] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*.

[34] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: intermittent execution without checkpoints. In *SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[35] Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. 2019. A Closer Look at Quality-Aware Runtime Assessment of Sensing Models in Multi-Device Environments. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems (SenSys '19)*. https://doi.org/10.1145/3356250.3360043

[36] Alessandro Montanari, Mohammed Alloulah, and Fahim Kawsar. 2019. Degradable Inference for Energy Autonomous Vision Applications. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp/ISWC '19)*. https://doi.org/10.1145/3341162.3349337

[37] Feng Nan and Venkatesh Saligrama. 2017. Adaptive classification for prediction under a budget. In *Advances in neural information processing systems*. 4727–4737.

[38] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: a Batteryless Long-Range Remote Visual Sensing System. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. 8–14.

[39] Sivert T Sliper, Oktay Cetinkaya, Alex S Weddell, Bashir Al-Hashimi, and Geoff V Merrett. 2020. Energy-driven computing. *Philosophical Transactions of the Royal Society A* 378, 2164 (2020), 20190158.

[40] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

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

[42] Zizhao Wang, Wei Bao, Dong Yuan, Liming Ge, Nguyen H Tran, and Albert Y Zomaya. 2019. SEE: Scheduling Early Exit for Mobile DNN Inference during Service Outage. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 279–288.

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

[44] Marco Willi, Ross T Pitman, Anabelle W Cardoso, Christina Locke, Alexandra Swanson, Amy Boyer, Marten Veldthuis, and Lucy Fortson. 2019. Identifying animal species in camera trap images using deep learning and citizen science. *Methods in Ecology and Evolution* 10, 1 (2019), 80–91.

[45] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5687–5695.

[46] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 285–300.

[47] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. ACM, 41–53.

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