Split Learning-based Sound Event Detection in Energy-Constrained Sensor Devices

Junick Ahn
Department of Computer Science
Yonsei University
Seoul, Korea
j.ahn@yonsei.ac.kr

Daeyong Kim
Department of Computer Science
Yonsei University
Seoul, Korea
dy.kim@yonsei.ac.kr

Hojung Cha*
Department of Computer Science
Yonsei University
Seoul, Korea
hjcha@yonsei.ac.kr

ABSTRACT

Sound event detection (SED) using lightweight sensor device has recently gained attention as a practical means to capture context and activities especially in domestic environments. However, SED applications running on sensor device are severely constrained by device's energy capacity. One solution is to offload a portion of inference to server for reducing runtime complexity, i.e., energy consumption, of sensor device. Offloading should consider the trade-off between computation and data transmission costs adequately; more computation on sensor device reduces data to be transmitted and vice versa. To address this challenge, we propose SEDAC (Sound Event Detection with Attention-based audio Compression), a novel technique for split learning in SED that compresses data from sensor device to offload less data. SEDAC compresses the input of SED models, or Mel spectrograms, with minimal computation in sensor device. Rather than directly compressing the input, SEDAC achieves data compression by selectively capturing the key parts of sound events using an attention mechanism. The scheme also modifies an existing loss function and employs knowledge distillation to mitigate potential loss of SED accuracy due to data compression. Our evaluation shows that SEDAC outperforms the state-of-the-art data compressive split learning schemes, up to about 30%. Furthermore, our real-world deployment demonstrates that sensor devices with SEDAC successfully operate with minimal energy and memory overhead.

KEYWORDS

Sound event detection, Model compression, Split learning, Sensor application, Energy-efficient system

1 INTRODUCTION

In recent years, sound event detection (SED) has received a great deal of attention as a method to obtain context and activities in target environments [1-3]. Many works have proposed models or training methods to detect events with sounds, especially in domestic environments [4-7]. To conduct SED in real environments, in-place sensor devices must record sound in their surroundings as audio data. Such sound sensors are often placed in multitudes to cover a large area.

Unfortunately, deploying sensors in appropriate locations is cumbersome or even infeasible with stationary wall-power. Thus, a location-agnostic wireless power source, such as harvested energy or a battery, is needed for the device, even though it has inherent energy restraints. SED application demands non-trivial operational energy for two reasons. First, SED requires sensor to sample and store audio, which consumes a large amount of energy even with minimal hardware. Inherently, this operation is mandatory in SED applications and can hardly be optimized. Second, in SED, the raw audio data need to be processed efficiently and effectively for event classification. Depending on the implementation, sensor device can simply transmit the raw data to a server to offload all computation, or process the audio data locally to reduce the transmission cost. The trade-off between computation and transmission cost is critical in optimizing energy cost of sensor device.

Optimizing energy cost of sensor device for SED applications relies on efficient handling of computation and transmission trade-off. One way of reducing energy cost is to minimize transmission overhead by employing a local inference model opted for sensor device. When using a local model for sound event detection, the device only needs to report event class. The solution minimizes transmission costs but increases computation overhead on sensor device, while resulting in reduced accuracy due to compressed models. A more promising solution is to exploit split learning method [8-9]. Split learning divides an original machine learning model into a subset of model running in sensor device and the rest of the model running at the server. This approach enables the sensor device to perform partial inference, while still benefiting from utilizing a sophisticated model for SED applications.

To optimize energy cost of split learning even further, *data compression* can be induced from inserting bottleneck layers. The bottleneck layers are inserted between the partitions which *compress* the intermediate output of sensor device. Compressing intermediate output of the partitioned layer is particularly effective for many sensor applications running on resource-constrained sensor devices [10]. While additional layers increase the computation cost slightly, data compression can reduce the

* Corresponding author.

979-8-3503-6201-5/24/$31.00 ©2024 IEEE
DOI 10.1109/IPSN61024.2024.00019

transmission cost significantly. To the best of our knowledge, bottleneck-based split learning has not yet been proposed for SED applications. Existing methods that allow data compression on different input types including image, or even raw audio, pose a problem for SED applications. Unlike other applications, SED application takes compressed data as input. In most state-of-the-art SED models, the input is Mel spectrogram or Mel-frequency Cepstral Coefficient (MFCC) [11]. The inputs are obtained by applying a short-time Fourier transform on raw audio data. Inherently, such transformed data are similar to those that are compressed using compressive sensing techniques. Compressive sensing technique achieves data compression through shifting domains and introducing sparsity to the data. As such, split learning models that utilize compressive sensing [12] may not be able to compress data further in SED applications.

In this paper, we propose the SEDAC (Sound Event Detection with Attention-based audio Compression) technique. SEDAC allows data compression in SED models that use Mel spectrogram as input. The spectrogram is further compressed based on our observation that the audio frames containing the important event is a fraction of the total audio. We show that this is a valid claim especially in real environments. Taking advantage of this factor, SEDAC selects Mel spectrograms that are likely to contain an event using attention-based mechanism [13]. Then, only the selected audio, or the Mel spectrograms are sent to the server to handle the rest of the inference computation. Using audio-selection mechanism, SEDAC achieves data compression that is difficult to be achieved with existing split learning models.

Moreover, SEDAC achieves high accuracy from online training, which updates model even after deploying a sensor. After offline training, sensor devices are deployed in real environments. Once they are deployed, the devices are still able to update the model using real sensor data. Different from other split learning methods, SEDAC allows online training with minimal overhead. SEDAC lets server to perform data augmentation during online training, as the intermediate output still retains the form of original Mel input.

The contribution of our work is as follows:

- To the best of our knowledge, SEDAC is the first effort to encompass split learning for SED. SEDAC compresses data from Mel spectrogram inputs, which are already a compressed form of audio data. SEDAC achieves high SED accuracy compared to existing split learning methods.
- We show that SEDAC can be implemented in a sensor device and be deployed in real environments. Also, the online training during real deployment is shown to improve model accuracy.

2 BACKGROUND AND RELATED WORK

We describe the related work and background on SEDAC in two parts. First, we introduce models and training methods related to sound event detection. We then introduce works related to split learning models.

2.1 Sound Event Detection Models

Over the years, sound event detection has been a challenging task, seeking new models and training methods. One distinctive characteristics of the state-of-the-art models is that most of them utilize transformed audio input such as Mel spectrogram [14] or MFCC. These transformations are based on Short-time Fourier transforms, where audio data shifts its domain from time to frequency domain.

Current state-of-the-art SED models are largely Convolutional Recurrent Neural Network-based (CRNN) or transformer-based models [15]. Presently, CRNN is the mainstream architecture for SED models. The models are based on RNNs which require processing each frame or sequence of input data. Inherently, CRNN-based models tend to have high computational cost. While not as dominant as CRNN-based models, transformer-based models are also effective for SED.

While the two models process data differently, the input remains the same; they both receive fixed-length audio as input. Rather than receiving a continuous stream of audio, CRNN receives a fixed-length audio, executes convolution operation, and processes convolution output in sequence using the RNN network. With Transformer model, the input is attached with its positional encoding to give each frame the positional information. Then, the input is fed to an attention layer to find the relationship among the audio frames. In either CRNN or Transformer models, state-of-the-art SED models typically depend on audio-specific data augmentations [16, 17] to enhance their performance. These augmentations alter time frames or frequencies to diversify the input. Unfortunately, during online training, the computational overhead of these augmentation methods is carried by the sensor devices.

2.2 Split Learning

Split learning [8] is a machine learning technique that lets edge device carry the first few layers of a whole model and does the rest in the server. Split learning has less constraints than other existing methods of model compression, e.g., quantization [18] or knowledge distillation [19], which conducts local inference in edge device. A local inference for SED has also been proposed [20], which is extremely lightweight and suited batteryless devices. However local inference limits performance of SED, as we demonstrate in Section 3. On the other hand, split learning avoids local inference by offloading most computations to the server. This allows the edge device to compute only a fraction of the whole model and transmit the intermediate output rather than the raw input. As a result, split learning effectively reduces the overhead on edge device while still leveraging the potential of complex models.

When partitioning a model with split learning, the last layer to be stored on the edge device is called cut layer. When executing the model, the total energy cost of the edge device is the sum of computational cost from computing the initial layer to the cut layer, and the transmission cost for delivering the intermediate output to server. Minimizing the energy cost of edge device is difficult, as the computation cost and the transmission cost are in a trade-off relationship. To minimize transmission cost, the output of the cut

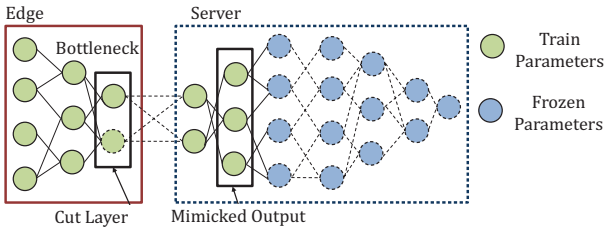


Figure 1. Example of bottleneck-injected split learning model [21]

layer has to be minimized, which generally requires deep layers with more computation. Shallower layers can reduce computational cost, but results in less compression, which increases output size of the cut layer.

To optimize the trade-off of transmission and computational cost of edge device, split learning models have recently exploited a bottleneck-injection method [21]. As shown in Figure 1, the method induces a bottleneck to compress the intermediate output. The output is then transmitted from an edge device to the server, reducing transmission cost. When compressing the intermediate output, the model may (1) compress the data up to a revertible state, or (2) compress them further to fit for a specific task, while making them irreversible in the process.

To achieve reconstructible compression, DeepCOD [12] has used a compressive sensing technique to the bottleneck layers. Compressive sensing is a signal processing technique that has been studied over a few decades. DeepCOD compresses data on an edge device and reconstructs them on the server. The fully reconstructed data allows different machine learning tasks to be used with its intermediate output. However, compressing data to a reconstructible level limits the data compression rate. More specifically, in SED task, the input already goes through signal processing using short-Fourier transform, which is similar to compressive sensing technique. Furthermore, achieving reconstructible feature requires the original input to be transmitted, as it is required to compute reconstruction loss. This incurs additional transmission overhead for edge device if an online training were to be performed.

On the other hand, data compression methods, such as BottleFit [22], achieves additional data compression specifically for a given machine learning task. To achieve such data compression, the entire encoding layers must be implemented on the edge device. These encoding layers tend to be deep layers, because shallow layers that induce high data compression make training more difficult [23]. In other words, higher data compression tends to force encoding layers to be deeper, which in turn increases computation overhead.

In summary, data compression with split learning is difficult, and may be even more difficult in SED applications. However, split learning for SED models can significantly improve the energy overhead of edge devices, as SED applications must record and process audio data frequently. Unfortunately, as far as we are aware, split learning on SED application has not been proposed yet.

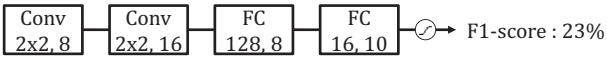


Figure 2. Simple model and its performance

3 PRELIMINARY EXPERIMENTS

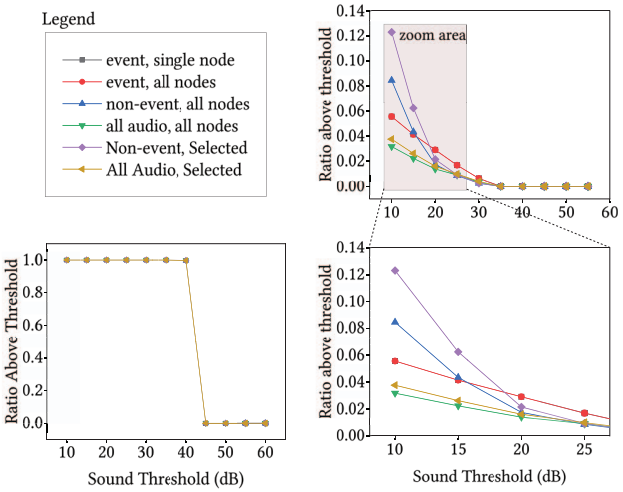
For SED application, edge device must record and process a large volume of audio data. Edge device can then either infer event class locally, or process data and send compressed data. Machine learning models impose large overheads to microcontrollers, in terms of computation and memory. Well-known parameter-efficient models [24-26] requires tens of megabytes of memory, if not hundreds. Simpler models, composed of a few convolution or linear layers, fit into microcontrollers, but are not sufficient for SED applications.

As a demonstration, we train a simple deep learning model, as shown in Figure 2, using the same preprocessing configurations described in Section 6 with the DESED dataset [27]. This model requires about 170KB RAM for its forward pass, which is higher than our proposed compression model on edge device. The F1-score of the model is significantly worse than the complex models shown in Section 6. Rather than implementing a full model in edge devices, the model can be offloaded a server. In this case, the model computation can be reduced through data compression can significantly reduce the total energy cost in a sensor device. One plausible way is to filter audio below signal power threshold. Unfortunately, based on our investigation, such audio preprocessing is inapt for SED.

To evaluate the audio filtering based on sound level, we examined the SINS dataset [28], a public dataset recorded in a real domestic environment for a week. When analyzing its annotated event labels, we exclude three event classes – absence, sleeping, and others event – that typically generate no or little sound associated with the events. When all the events are summed together, the duration of events amounts to about 53.37% of the total recordings.

Of the total duration of events, not all the audio signals are significant. Even the audio within labelled events may not carry signals that identifies the event. One method of selecting important signals is power level analysis, which typically requires audio preprocessing such as noise reduction. Figure 3 shows the ratio of captured audio in different sound level thresholds. The figure shows the ratio of significant power level above various thresholds, with respect to different audio recordings. As shown in Figure 3(a), without any preprocessing, the raw audio classifies all audio as significant when the threshold is equal to or less than 35 dB, and as insignificant when greater than 40 dB. This is due to the presence of high-level noise from the microphone. In sum, selecting event-containing audio is difficult due to the persistent microphone noise.

On the other hand, noise reduction process has the potential to eliminate microphone noise. However, this process is computationally intensive and lacks the ability to distinguish events from other ambient noises. In Figure 3(b), the change in the ratio of audio above a specified threshold is depicted when noise reduction is employed. The legend is denoted as a combination of "audio



(a) Without noise reduction (b) With noise reduction

Figure 3. Ratio of sound above threshold with different audio recordings for audio category and node type (audio type, selected node)

type" and "nodes." "Audio type" specifies whether the audio originates from a labeled event (event), from non-event sources (non-event), or encompasses all audio (all). "Nodes" indicates whether the selected audio is from a node corresponding to the specific room where an event occurred (selected) or encompasses all twelve nodes (all). In contrast to Figure 3(a), Figure 3(b) shows a gradual decrease in the ratio of audio above the threshold as the threshold value increases. However, the ratio of captured audio in (event) case is not higher than that in the (non-event) case. In fact, the captured audio ratio in the (non-event) case surpasses that in the event scenario, approximately 3.27 times more at a 10 dB threshold. In addition, noise reduction is computationally expensive. In our setup using the noise_reduction Python library, the execution time of sound threshold analysis was increased by fifteen times.

From the preliminary experiments, we suspect that an audio compression model designed for SED, as illustrated in Figure 4, can be an effective solution for an SED application. Rather than filtering out insignificant sound level, the model can learn to recognize useful signals from the audio. Then, the model can exclude hardware or environmental noise from the signal, compressing audio data as a result. Furthermore, the SED application does not have to sacrifice classification accuracy as an edge device can offload computation to a server.

4 DESIGN OF SEDAC

We state the problem and the goal of SEDAC. Then, we describe the design of SEDAC in three parts – audio selection, loss function, and training process.

4.1 Problem Statement

Capturing an event-containing audio is an important task that is difficult to be optimized on sensor device. As we have shown in

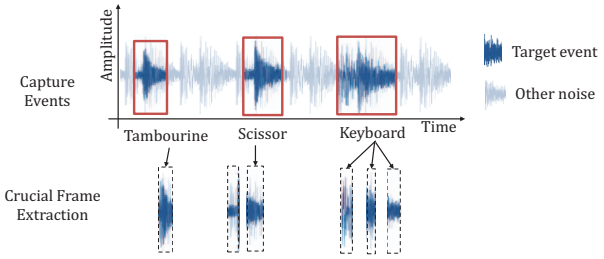


Figure 4. Example of audio-selective SED model

Section 3, local inference, or algorithms such as noise reduction is not ideal for the task. As such, we devise a split learning model which lets edge device compress the audio data and offloads computation to a server. Implementing a split learning model for SED sensor application must overcome two key challenges.

First, data compression from Mel spectrogram should be achieved effectively. Typical SED models take Mel spectrogram as inputs, which are the compressed form of raw audio. Data compression is still achievable through selecting key audio frames. As we investigated in Section 3, the key audio frames that represent target events are only a fraction of total audio recordings. Furthermore, the key audio frames are only a part of the total event audio. Figure 4 shows an example of capturing the key audio frames. The first process is to select the entire frames between the onset and offsets of each event. The following step is to pick only the frames necessary to classify the event. For many events such as the keyboard event shown in the figure, a repetition of the same signals is found over a long duration. In addition, even in one-time events, such as tambourine or scissor event in the figure, SED may only require a few key frames including peak sound signals to successfully classify the event type.

Second, the split learning model for SED should be constructed carefully. If a split learning model were to select only a portion of audio input to be transmitted to server, then a different loss function could function better; evaluating the already omitted audio frames may not be necessary or be ineffective for training. Furthermore, a post-deployment online learning scheme can improve the split learning model. Similar to other audio processing models, SED application is affected by audio variability due to hardware or environment factors. In such cases, the pre-trained model trained during the offline learning may not be optimal for the target environment. Split learning allows continual learning after deployment as an online process. However, the system should update model efficiently because updating a model requires additional overhead.

4.2 System Overview

The overview of SEDAC is shown in Figure 5. SEDAC has a two-stage process. The first stage is the offline training, and the second state is an online training after deployment. The key component of SEDAC, data compression through audio selection, is based on an attention probability matrix. This component is present during both the offline and online processes. During the offline process, data augmentation is applied prior to data compression. Data

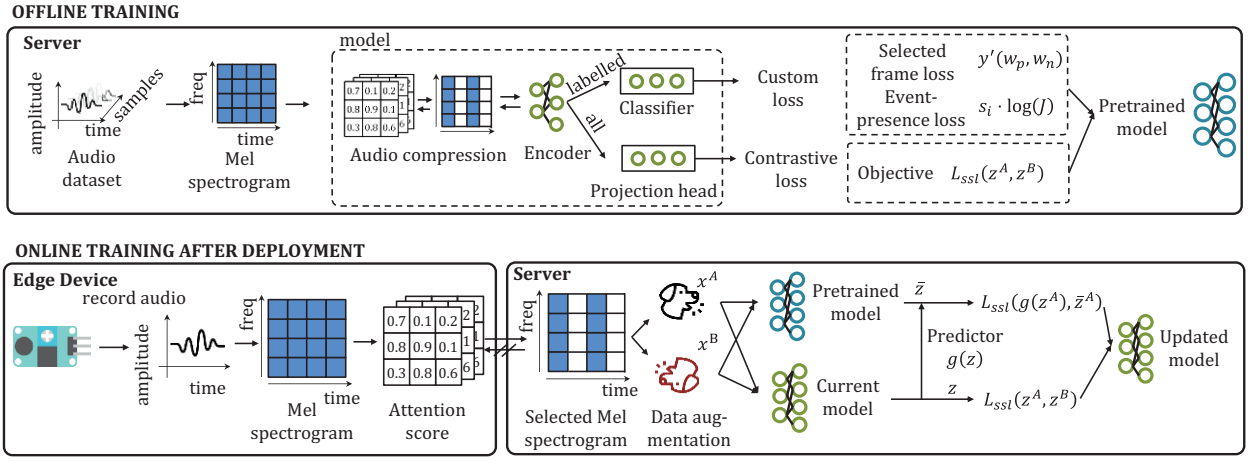


Figure 5. Overview of SEDAC

augmentation is applied after data compression during online training. This reduces computation overhead on the sensor-side while making online learning effective. After data augmentation and compression, the selected audio data is processed further in the classification model. The output of the model, i.e., the selected event for each frame, is evaluated with a modified version of a binary cross entropy loss function.

4.3 Attention-based Audio Selection

In split learning, reducing intermediate output from the cut layer is crucial in reducing transmission cost of sensor device. SEDAC compresses the output based on event-based audio selection. The audio selection in SEDAC not only omits audio not containing an event, but also selects the key frames within the whole event occurrence. To achieve such data compression, SEDAC proposes an audio selection scheme based on relative multi-headed self-attention mechanism [15].

Attention mechanism has been employed in many state-of-the-art machine learning models. The mechanism essentially reads sequential data and converts them into a representation which maps the importance or relevance of a feature to the rest of the features. Using this mechanism, we propose an algorithm that determines the most important features in a given audio sequence.

Figure 6 shows the attention-based compression algorithm in SEDAC. First, the input sequence, or the frame dimension, is pooled to a smaller size; each feature from the frame dimension now represents a larger window. The pooling reduces the number of sequences such that the model can infer the relevance between the sequences easily. After pooling, the features are passed on to the relative multi-headed self-attention layer. The layer encodes the given input with a relative positional embedding. Then, attention is computed with a multi- headed attention where there are multiple sets of key matrices K, query matrix Q, and value matrix V. The input dimension of the self-attention is frame dimension rather than Mel or frequency dimension, because we want to select audio frames from the time axis.

The output of our self-attention layer is the attention probability, rather than the regular output. This excludes the use of

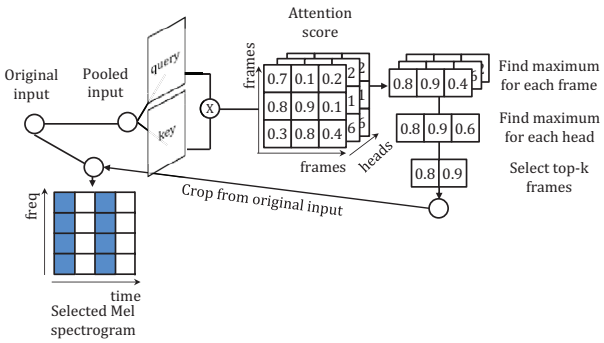


Figure 6. Attention-based audio selection mechanism

value matrix, as shown in Figure 6. The attention probability matrix is calculated as $Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$, where d_k is dimension of key matrix. Next, we convert the attention probability into the probability to select a given frame of matrix. To do so, we find maximum values of the given matrix along the frame dimension. Intuitively, this computes the most important frame with respect to each sequence. Then, we find maximum values along the head dimension, to pick the most relevant frame among different attention probability maps. The output matrix after these steps is now a single probability score for every sequence. The scores are filtered once again by a top-k operation to select the k number of highest score sequences. The k is selected such that the ratio of k and the number of features in pooled input are equivalent to the desired compression ratio. Then, the indices of k are inflated back to match the size of the original input to select audio frames from the original input. Finally, based on the selected sequence indices, the audio frames of the original input sequence are chosen. Figure 7 illustrates an example of the audio selection process. This audio selection layer is set as the cut layer of SEDAC, upon which the output is transmitted to the server where the rest of the model is computed.

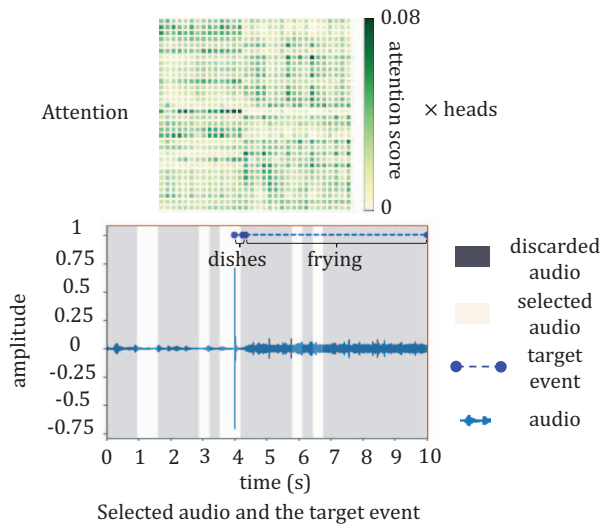


Figure 7. Attention-based audio compression process

4.4 Loss and Evaluation Function

Different from the existing SED models, SEDAC does not determine the exact onset and duration of a given event. Because SEDAC makes the model discard information on many audio frames, the evaluation function or the loss function has to be adjusted accordingly. First, we state the criteria for correct output in SEDAC as follows:

1. Does the output contain all the events in the audio data? To do so, a single audio frame must be selected at least for each of the event given.
2. Are all the selected frames lie within the boundary of onset and offset of the event?

Based on these two criteria, we compute F1 score to evaluate a SED model.

Similarly, based on the criteria, we also modify a binary cross entropy loss function for SEDAC. The modified loss function does not penalize misclassification of unselected frames. Instead, SEDAC introduces event-presence loss, which penalizes selecting audio frames that do not contain any event. In summary, the modified loss function of SEDAC is defined as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N} w_p \cdot y'_i \cdot log\big(p(y'_i)\big) + w_n \cdot \big(1 - y'_i\big) \cdot log\Big(1 - p\big(y'_i\big)\Big) + w_s \cdot s_i \cdot log(J) \quad (1)$$

Where y' is the selected frames, w the weight, and J a matrix of ones. Term s_i is the event presence matrix, which sets value 1 to frames that contain one or more events, and sets 0 for frames that do not. Figure 8 visualizes the three terms in our modified binary cross entropy loss function. The top graph shows the event class and time label (ground truth), and the model prediction. The model prediction is in two parts, the selected audio part (selected frame), and class prediction within the selected audio part (selected event). The bottom graph indicates where loss would occur, given the model's prediction and ground truth label. As shown in the bottom

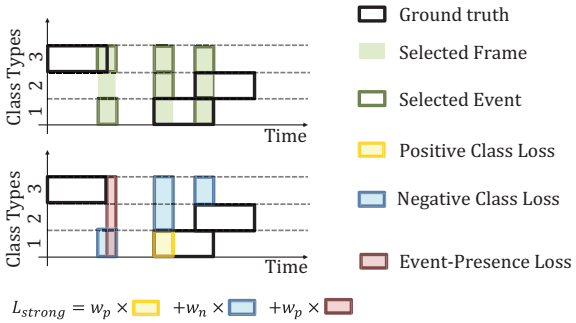


Figure 8. Example of loss evaluation in SEDAC

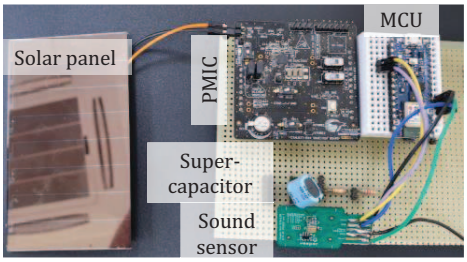
figure, a typical SED task contains more negative classes than the positive ones. As such, negative class loss is likely to be greater than the positive class loss, and thus we set the positive class loss w_p to be greater than the negative class loss w_n to avoid negative class losses dominating the loss value. Last, the event-presence loss guides the model to select audio frames within the audio boundaries of an event. Note that the weak loss function, or the classification-only loss function, returns the same result for evaluating either all frames y or selected frames y', and cannot have event-presence loss term. Thus, the loss function needs not be modified, applying only the positive and negative class weight terms.

4.5 Model Training

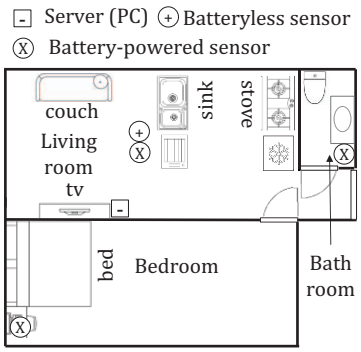
SEDAC trains an SED model with both offline and online training. An SED model is trained in a server during an offline training. During offline training, as shown in Figure 5, all data from the dataset are used for contrastive learning [29], whereas the data with labels are used to train the model with a classifier layer. Essentially, contrastive learning uses the data without label to output a feature vector through a projection head layer. Supervised learning uses labelled data to output a classification vector through a classification layer. Both learnings share the same features except the final layer.

We use contrastive learning for both online and offline training to train the model without labels. Contrastive learning operates by optimizing loss L_{ssl} between the same sample with different data augmentations, z^A and z^B, such that $L_{ssl}(z^A, z^B)$ is minimal. We use Barlow Twins [32] for loss function L_{ssl}, which uses correlation-matrix in the loss function. The loss function is defined as $\Sigma_u(1 - \mathrm{C}_{uv})^2 + \lambda\Sigma_u\Sigma_{u\neq v}C_{uv}^2$, where C_{uv} is the correlation-matrix defined as $\mathrm{C}_{uv} = \frac{\Sigma_i z_{i,u}^A z_{i,v}^B}{\sqrt{\Sigma_i (z_{i,u}^A)^2}\cdot\sqrt{\Sigma_i (z_{i,v}^B)^2}}$.

After deployment, online training is conducted. One benefit of online learning after deployment is to adapt to the hardware and environment of the deployed sensor. As previously shown, the input data may vary due to different hardware setups. Such variability negatively impacts the result of sound related machine learning [30]. SEDAC allows online learning with minimal overhead to adapt to the real environment of the deployed sensor device.



(a) Batteryless hardware



(b) Home environment

Figure 9. (a) Hardware setup and (b) the deployment environment for SEDAC

Different from an offline training, online training of SEDAC allows any data augmentation techniques, e.g., SpecAug, to be applied after audio selection, as original input form is maintained. Furthermore, the model update is only done within a server, and the audio selection layers in the edge device are not updated further. In online training, training data is gathered from sensors. Because the data have no labels, our online learning is performed solely through continual contrastive learning [31]. The method uses self-supervised contrastive learning as a self-distillation mechanism to update the model. Both the pretrained and the online model has the same structure, with the only difference in having a predictor layer g in the online model. The online model inherits the parameters from the pretrained model. During training, the output of online model without predictor layer g is first compared with two different augmentations, denoted as $L_{ssl}(z^A, z^B)$. Subsequently, the output through g is compared to the output of the pretrained model, expressed as $L_{ssl}(g(z^A), \bar{z}^A)$, where $\bar{z}$ is the output of a pretrained model obtained from offline training. The final loss of the online learning is then defined as $L_{ssl}(z^A, z^B) + L_{ssl}(g(z^A), \bar{z}^A) + L_{ssl}(g(z^B), \bar{z}^B)$. Note that the online training exclusively updates the representations, and not the classifier. The classifier needs separate training.

5 IMPLEMENTATION

The training process and the SEDAC model are built using TensorFlow. The model input, a Mel spectrogram, is extracted using 1024 window size, 626 hop length for short-time Fourier transform. The experiments in Section 6.1 uses 16 kHz audio sample. Audio preprocessing and the extraction of Mel spectrogram from the data is performed using Librosa [33], a Python library for sound processing.

The model trained for experiments in Section 6.2 uses 2.5 seconds 8 kHz audio, due to limitation in memory of the target edge device. The target edge device is Arduino Nano 33 BLE microcontroller, with 64 MHz clock and 256 KB SRAM. As sound sensor, we use a VM1010 wake-on-sound microphone. The system is put into shutdown until it receives an interrupt signal from the wake-on-sound microphone sensor. The audio is sampled and converted to Mel simultaneously, using a buffer. The spectrogram is then passed to the split learning model. The model is pretrained and then converted to a TensorFlow Lite model. The intermediate output from the cut layer is transmitted to the laptop server using BLE.

Batteryless version of SEDAC includes amorphous solar panel [34] as a power source and a 0.1F-5V supercapacitor [35], as shown in Figure 9(a). The microcontroller shuts down whenever sound is not present. The system reboots when an interrupt signal is received from the wake-on-sound sensor.

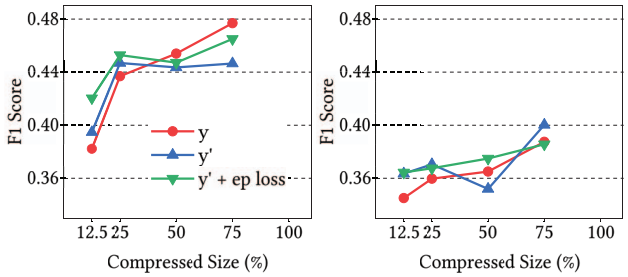
6 EVALUATION

We evaluate SEDAC in three different aspects. First, we apply SEDAC on two state-of-the-art models, FDY-CRNN [36] and ConformerSED [37], to examine the impact of data compression. The models with data compression are compared with the ones without any data compression, to observe the accuracy loss when data compression is enforced in those models. Second, we compare SEDAC with two existing data compression models, DeepCOD [12] and BottleFit [22]. All three compression models are modified, such that the intermediate features have the same compression rate. Third, we implement SEDAC on a sensor device and deploy it in a real environment. We observe how SEDAC in a real environment reduces energy cost and improves model accuracy through online learning.

6.1 SEDAC in State-of-the-Art Models

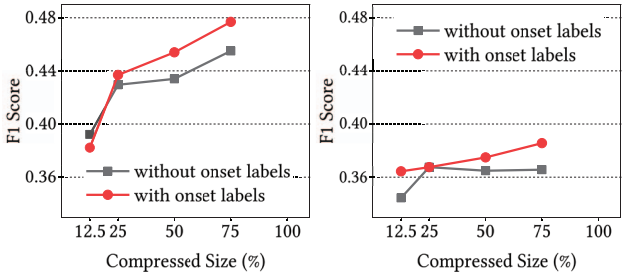
Overall performance We select two state-of-the-art models, ConformerSED and FDY-CRNN, to implement SEDAC. ConformerSED uses a convolutional Transformer network, whereas FDY-CRNN employs an CRNN architecture. We modify both models to receive the same Mel input. The dataset used for training is the DESED dataset [27] which contains synthetic, weak, unlabeled, and validation data. The synthetic dataset mixes the audio with background noise from other sources. The synthetic dataset contains both the event class label and the event onset and offset label. The weak dataset only contains the event class label, whereas the unlabeled dataset does not contain any target label. The validation dataset contains audio extracted from YouTube, with both the event class label and the onset and offset label.

Using this setup, we train two state-of-the-art models with different compression rates. First, during training without data compression, the model is trained using both regular binary cross entropy loss and weighted binary cross entropy loss with 1.0 weight to positive class and 0.1 to negative class. For model trainings with



(a) ConformerSED (b) FDY-CRNN

Figure 10. F1 score of SEDAC-applied models on different state-of-the-art models (best out of 10 trainings)



(a) ConformerSED (b) FDY-CRNN

Figure 11. F1 score of SEDAC by different compression rates when trained with weak-label data only (best out of 10 trainings)

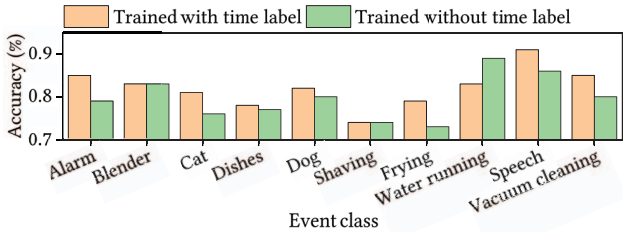


Figure 12. Audio-selection accuracy of SEDAC trained with and without time label data

data compression, we test each case, where (1) weight is applied to the loss function, (2) loss is computed only against the selected frames, and (3) event presence loss term is added.

The accuracy comparison of different models is shown in Figure 10, where y denotes a regular loss function, y' a modified loss function without event-presence loss term (eploss), and y' + eploss is modified loss function with event-presence loss term. In terms of different loss functions, the modified loss function tends to yield higher model accuracy than that of the regular loss function. Specifically, in models with smaller compressed sizes, the difference seems to be significant. When the compressed data sizes are 25% or 12.5%, the accuracy difference between the regular loss function and the modified loss function is significant, between 3% to 5% difference, in both ConformerSED and FDY-CRNN models.

The regular loss function tends to show similar or better results in larger compressed data sizes. This is an expected result, because the modified loss function is designed to penalize irrelevant selection of the audio segment. With larger compressed sizes, SEDAC discards fewer audio data, making selection of irrelevant audio unavoidable. The comparison indicates that our modified loss function, which does not penalize the discarded audio data, yields better results, especially when data compression rate is high.

Note that both models show poor training results without weighted loss function, and thus are excluded in the figure. In summary, we observe that SEDAC shows little or no decrease of model accuracy when compression is applied.

Learning without time label SEDAC inserts a layer that induces audio selection. While it may seem the audio selection relies on onset and offset label data, audio selection based on event can be learned only with the weak label, or the event class-only label. To demonstrate learning solely from weak-label data, we train SEDAC without any onset and offset label; the onset and offset label are omitted in the synthetic dataset. After training, we use the validation dataset to test whether the audio selection layer selects event-containing audio. As shown in Figure 11, while being slightly worse than the fully trained models, training solely with weak-label data can achieve similar F1-score.

In addition, we examine the accuracy of audio selection when learning without time labels. As shown in Figure 7, audio selection uses attention score matrix to extract the indices of the selected audio frame. Rather than the final output, we compare the selected indices from the compression layer with the ground truth time label. The performance of audio selection mechanism is evaluated as follows. We set positive case as when the selected audio is within the boundaries of an event. The negative case is when no frames of the selected audio lie within the event boundaries. When multiple events occur within the 10s audio clip, every event is tested. We evaluate the performance of SEDAC model with 25% compressed data size. The result is shown in Figure 12. The average accuracy of audio selection mechanism in model trained with time label data is 84.7%, whereas the average accuracy of model trained without time label is 81.1%. Hence, while not as accurate, the latter trained model achieves similar accuracy to the former trained model.

Our experiments suggests that the audio compression layer can determine important audio segments without any ground-truth time label. This allows model training with any public audio datasets without time labels.

Comparison with other compressive models We investigate whether the existing compressive models operate effectively in SED, and verify if SEDAC outperforms the existing compressive models in SED. The state-of-the-art compressive models include DeepCOD and BottleFit. DeepCOD is a technique that utilizes a level of compression capable of reconstruction, while BottleFit compresses features further, which are not reconstructible anymore.

The performance differences with each model are shown in Figure 13. First, DeepCOD shows a decline in performance as the compression rate increases. This occurs in all metrics; F1 score, recall, and accuracy. The degradation is due to the additional compression induced on data that is already compressed using Fourier transform. Particularly, the magnitude of performance decline becomes larger as the compression rate increases. Similarly,

Table 1. Current consumption of SEDAC tasks in Arduino

Sampling		Mel	Mel + Sampling (per frame)		Audio Selection		Transmission		Standby
current (mA)	Time (s)	current (mA)	current (mA)	Time (ms)	current (mA)	Time (ms)	current (mA)	Time (ms)	Current (uA)
0.82	2.5	3.34	2.84	56	4.13	42	8.77	101	40

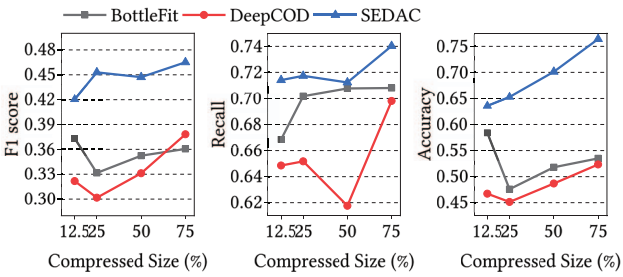


Figure 13. Performance comparison between different compressive models

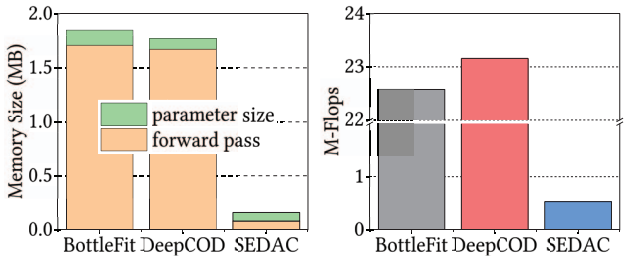


Figure 14. Memory (left) and computation (right) cost of different compressive models in edge device

BottleFit performs slightly better than DeepCOD, but shows significantly less F1 score and accuracy than SEDAC. SEDAC shows the highest performance in all compression rates.

6.2 SEDAC on Edge Devices

Overhead On edge devices, only the compression layers are installed. We examine the memory and computation cost of different compressive models in Figure 14. SEDAC exhibits significantly smaller memory and computation costs compared to BottleFit and DeepCOD, being approximately 13 times less in both memory (166 KB) and computations. This substantial difference stems from the design of the encoder layers. SEDAC strategically selects useful audio data with lightweight model and defers encoding to the server. In contrast, BottleFit and DeepCOD assume that edge devices possess more robust computational capabilities and larger memory sizes, such as those in mobile phones. Conversely, SEDAC targets low-power microcontrollers as its intended devices. The microcontroller-optimized version of SEDAC, described in Section 5, only requires about 76 KB of Flash memory and 32 KB of RAM.

Given relatively small footprint and model computation, we explore the possibility of constructing SEDAC as a batteryless application, constructed as described in Section 5. The total energy cost of SEDAC on edge devices consist of four tasks: (1) audio

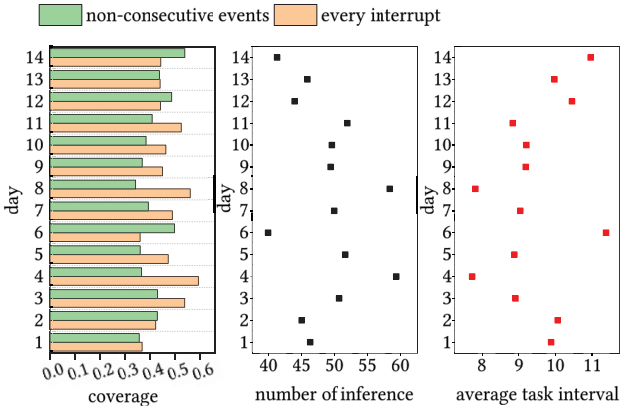
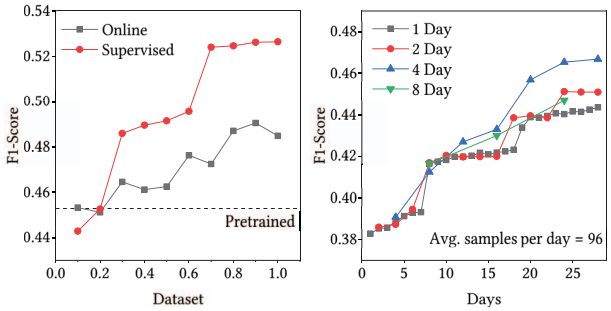


Figure 15. Performance of SEDAC on a batteryless device

sampling, (2) Mel calculation, (3) audio compression and (4) transmission. Tasks (1) and (2) largely occurs simultaneously, while the other tasks run consecutively. The current consumption for each task, measured using a digital multimeter, is presented in Table 1. The execution time of each task are 2500, 56, 42, and 101 milliseconds, respectively, with the total energy consumption of 27.08 mJ. For 16kHz, 10s sampling, the total energy cost is 173.8 mJ. The audio compression computation only costs 1.92% of the total energy consumption while saving transmission cost.

SEDAC also saves energy cost compared to two other plausible methods, sampling and sending raw data, or computing and transmitting Mels. We test these scenarios and compare the energy consumption in 8 kHz, 2.5 s sampling and 16 kHz, 10 s sampling setups. First, sending raw data requires sending x10 more data in 8kHz setup, and about x40 more data in 16 kHz setup. As a result, transmission time increases to 1001 ms and 7992 ms respectively, resulting in total of 35.8 mJ and 296.5 mJ. In other words, sending raw data costs 32% and 71% more energy than using SEDAC. The efficiency of SEDAC increases with a larger sample size when compared to transmitting raw data. Second, sending Mels also costs more energy than SEDAC, as the compression costs less energy than transmission. In 8 kHz setup, transmission time was about 411 ms, resulting in 35.48 mJ or 31.0% increase in energy cost. In 16 kHz setup, transmission time was about 820ms, resulting in 190.70 mJ or 9.7% increase in energy cost. The difference of energy cost between Mel transmission and SEDAC is smaller as the sample size grows; when the sample size is larger, the ratio of Mel computation cost to transmission cost becomes higher.

Using our setup, we deploy a batteryless device and a battery-powered device over two weeks in a living room. As shown in Figure 9(b), the batteryless sensor and the battery-powered device are placed side-by-side to allow recording the same audio data. The



(a) DESED Validation dataset (b) Real deployment data

Figure 16. Performance improvement through online learning

result of the batteryless application deployment is shown in Figure 15. When compared to the battery-powered device, batteryless device detects about 47% of the total events on average. For non-consecutive events, i.e. 100 events detected from a 10-minute vacuum cleaning event, are treated as a single event, the average coverage of the events is 41.6%. This is due to the occurrence of long consecutive events where sound interrupts occur continuously on the same event. In days where there are more long consecutive events, i.e. day 4 and 8, the average coverage counting all interrupts tend to be high, but non-consecutive coverage tends to be low. In such cases, due to many consecutive events, the average task interval tends to be short, and the number of inferences tends to be high.

Online learning We first conduct an online learning experiment for SEDAC using the DESED validation dataset to simulate an online training. We remove the labels from the validation dataset. Then, the dataset is shuffled and split into ten datasets to be loaded incrementally; for every 100 epochs, the model is trained with additional 10% of the dataset. Then, we examine if the online learning can improve model performance without label data. As baselines, we compare the results with supervised offline learning and pretrained model. Supervised offline learning is given the label data, and a separate model is trained for every different incremental dataset. The classifier layer of online model is trained separately, with the label data. The pretrained model uses the model trained in Section 6.1.

Figure 16(a) shows the result of our experiment. Contrastive online learning shows continuous improvement of F1-score over the pretrained model. The learning achieves about 49% F1-score in the later stages, which shows 3.2% improvement over the pretrained model. On the other hand, the difference between the contrastive online learning and the supervised learning also remains significant, reaching 3.56% at ninth incremental dataset.

Next, we examine the online learning we conducted in a real deployment scenario. We deployed three battery-powered Arduino devices with VM1010 sensors over four weeks. We place one sensor each in a living room, a bedroom and a bathroom, as shown in Figure 9(b). We manually label the data we collect by letting the sensor also transmit the original audio, to train the classifier layer. The compressed audio from all three sensors is transmitted to a server, or a PC, which is accumulated incrementally over the deployment period. We use the pretrained model as our model obtained during offline training, where we use 8 out of 10 target events – alarm bell, blender, dishes, toothbrush, frying, running water, speech, and vacuum cleaner event.

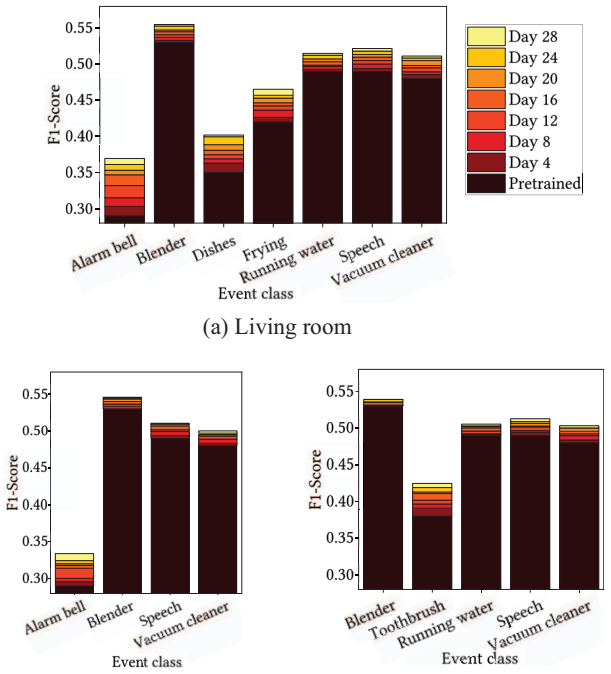
First, we train models using all three nodes. We train four different models, where each wait 1, 2, 4, or 8 days to accumulate an epoch of training data. The models are then evaluated using all the samples collected at the end. The change in F1-score of SED tasks in real deployment experiment is shown in Figure 16(b). The highest F1-score was achieved with a 4-day model. As we collected 96 samples per day on average, each increment to the dataset of 4-day model was about 384 samples on average. Models with smaller increments, e.g. a 1-day and a 2-day model, show less F1-score. The result indicates that more frequent training, with smaller increments to the dataset, does not always yield the best result. On the other hand, training intervals that are too infrequent, e.g. 8-day model, also yield suboptimal results. In all scenarios, online learning significantly improved F1-score on classification over time.

Second, we train model separately for each location. The samples are accumulated in 4-day interval. In each location, different classes of events occur. In the living room, a total of 7 event classes occurred, alarm bell, blender, dishes, frying, running water, speech, and vacuum cleaner event. In the bedroom, a total of 4 event classes occurred, alarm bell, blender, speech, and vacuum cleaner event. In the bathroom, a total of 5 event classes occurred, blender, toothbrush, running water, speech, and vacuum cleaner event. Note that blender, vacuum cleaner event, and speech events may occur in one location but may be recorded in other locations as well. We evaluate the trained models with all the samples from all three locations. As shown in Figure 17, all online trainings show noticeable improvements over the pretrained model. Generally, the lower the F1-score is in the pretrained model, the improvement is more significant. In addition, the trained model from living room shows the highest F1-score. We suspect that this is due to having higher number of samples. Even if there were same number of samples within the same class, having more samples from other classes helps contrastive learning due to having more negative samples.

7 CONCLUSION

SED in a real environment requires energy efficient operations of sensor device. SEDAC achieves the goal through split learning and data compression. SEDAC extracts only the relevant portions of audio where sound events are expected. As such, SEDAC can achieve additional data compression from Mel spectrograms. Moreover, SEDAC retains the Mel format after compression, and thus is effective for applying various training techniques in the server during online training.

SEDAC primarily focuses on the efficient server offloading aspect of split learning, refraining from training model parameters on the edge device. Despite the potential performance improvement from parameter training on edge device, the existing library, TensorflowLite-Micro, does not support such updates. In addition, SEDAC focuses on fine-tuning a model to the target environment.



(a) Living room

(a) Bedroom

(c) Bathroom

Figure 17. Online learning conducted in each node in three different locations

Building an aggregation of multiple personalized models for various sensor hardware and environments can be another problem that can be addressed in the future.

Another area of future work is on continual online learning. SEDAC can learn representations through contrastive learning and distillation, but the classifier layer still has to be supervised separately. The classifier layer may be learned unsupervised with clustering methods, but training classifiers with predefined classes is a challenge to be solved in online continual learning. In addition, class-incremental learning, a scenario where new event classes might emerge during online learning is being discussed little in SED. We believe that this is another important topic in SED.

ACKNOWLEDGEMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00532, Development of High-Assurance (≥EAL6) Secure Microkernel)

REFERENCES

[1] Bashima Islam, Md Mahbubur Rahman, Tousif Ahmed, Mohsin Yusuf Ahmed, Md Mehedi Hasan, Viswam Nathan, Korosh Vatanparvar, Ebrahim Nemati, Jilong Kuang, and Jun Alex Gao. 2021. BreathTrack: Detecting regular breathing phases from unannotated acoustic data captured by a smartphone. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–22. https://doi.org/10.1145/3478123

[2] N. C. Phuong and T. D. Dat, Sound classification for event detection: Application into medical telemonitoring. 2013. In *Proc. Int. Conf. Comput., Manage. Telecommun*. (ComManTel), Ho Chi Minh City, Vietnam, Jan. 2013, pp. 330–333.

[3] S. Grollmisch, J. Abeber, J. Liebetrau, and H. Lukashevich. 2019. Sounding industry: Challenges and datasets for industrial sound analysis. In *Proc. 27th Eur. Signal Process. Conf. (EUSIPCO)*. A Coruna, Spain, Sep. 2019, pp. 1–5.

[4] E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 25, no. 6, pp. 1291–1303, Jun. 2017.

[5] Chen, Y. and Jin, H. Rare Sound Event Detection Using Deep Learning and Data Augmentation. 2019. In *Proceedings of the 20th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. 619-623.

[6] Koh CY, Chen YS, Liu YW, Bai MR. 2021. Sound event detection by consistency training and pseudo-labeling with feature-pyramid convolutional recurrent neural networks. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021 Jun 6 (pp. 376-380).

[7] Chen, Y. and Jin, H. Rare Sound Event Detection Using Deep Learning and Data Augmentation. 2019. In *Proceedings of the 20th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. 619-623.

[8] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems.* 615–629. https: //doi.org/10.1145/3037697.3037698

[9] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2021. Split computing and early exiting for deep learning applications: *Survey and research challenges. Comput. Surveys* (2021), 1–28.

[10] Bakhtiarnia A, Milošević N, Zhang Q, Bajović D, Iosifidis 2023. Dynamic split computing for efficient deep edge intelligence. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1-5.

[11] Beth Logan. 2000. Mel frequency cepstral coefficients for music modeling. In Proceedings of the International Symposium on Music Information Retrieval (ISMIR'00).

[12] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 476–488.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'15)*. http://arxiv.org/abs/1409.0473.

[14] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerrv-Ryan, et al. 2018. Natural TTS synthesis by conditioning WaveNet on Mel spectrogram predictions. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 4779–4783.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In NeurIPS, Los Angeles.

[16] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. 2019. SpecAugment: A simple data augmentation method for automatic speech recognition. In *Proceedings of the 20th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. 2613–2617.

[17] Hyeonuk Nam, Seong-Hu Kim, and Yong-Hwa Park. 2022. Filteraugment: An acoustic environmental data augmentation method. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4308–4312.

[18] Y. Zhou, SM. Moosavi-Dezfooli, NM. Cheung, and P. Frossard. Adaptive quantization for deep neural network. 2018. *Thirty-Second AAAI Conference on Artificial Intelligence*.

[19] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization, and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7130–7138.

[20] Monjur M, Luo Y, Wang Z, Nirjon S. 2023. SoundSieve: Seconds-Long Audio Event Recognition on Intermittently-Powered Systems. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 28-41.

[21] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. 2019. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges*. 21–26.

[22] Matsubara, Yoshitomo, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing. 2022. In *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 337-346.

[23] Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810 (2017).

[24] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning (PMLR)*. 6105–6114.

[25] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. 2017. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *31st AAAI Conference on Artificial Intelligence*.

[26] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7132–7141.

[27] Serizel R, Turpault N, Shah A, Salamon J. Sound event detection in synthetic domestic environments. 2020. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 4 pp. 86-90.

[28] Dekkers G, Lauwereins S, Thoen B, Adhana MW, Brouckxon H, Van den Bergh B, Van Waterschoot T, Vanrumste B, Verhelst M, Karsmakers P. The SINS database for detection of daily activities in a home environment using an acoustic sensor network. 2017. *Detection and Classification of Acoustic Scenes and Events 2017*. 1-5.

[29] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 3733–3742

[30] Akhil Mathur, Anton Isopoussu, Fahim Kawsar, Nadia Berthouze, and Nicholas D Lane. 2019. Mic2mic: Using Cycle-consistent Generative Adversarial Networks to Overcome Microphone Variability in Speech Systems. In *IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[31] E. Fini, V. G. T. da Costa, X. Alameda-Pineda, E. Ricci, K. Alahari, and J. Mairal. 2023. Self-supervised models are continual learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 9621–9630.

[32] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow Twins: Self-supervised learning via redundancy reduction. In *Proceedings of the International Conference on Machine Learning* (ICML'21). 12310–12320

[33] librosa. Retrieved June 29, 2023, from https://librosa.org/doc/latest/index.html

[34] AM-1816CA. Retrieved Oct 17, 2023, from https://www.mouser.com/datasheet/2/315/panasonic_AM-1816CA-1196985.pdf

[35] Eaton PM Supercapacitors. Retrieved Oct 17, 2023, from https://www.mouser.com/datasheet/2/87/eaton_pm_supercapacitors_cylindrical_pack_data_she-1608781.pdf

[36] Hyeonuk Nam, Seong-Hu Kim, Byeong-Yun Ko, and Yong-Hwa Park. 2022. Frequency dynamic convolution: Frequency-adaptive pattern recognition for sound event detection. *arXiv preprint arXiv*:2203.15296.

[37] Miyazaki K, Komatsu T, Hayashi T, Watanabe S, Toda T, Takeda K. 2020. Conformer-based sound event detection with semi-supervised learning and data augmentation. *in Proc. Detection Classification Acoust. Scenes Events Workshop*. pp. 100–103.