IMPLICIT ACOUSTIC ECHO CANCELLATION FOR KEYWORD SPOTTING AND DEVICE-DIRECTED SPEECH DETECTION

Samuele Cornell[1*], *Thomas Balestri*[2], *Thibaud Sénéchal*[2]

[1]Università Politecnica delle Marche, Italy
[2]Amazon Alexa AI, USA

ABSTRACT

In many speech-enabled human-machine interactions, user speech can overlap with the device playback audio. In these instances, the performance of tasks such as keyword-spotting (KWS) and device-directed speech detection (DDD) can degrade significantly. To address this problem, we propose an implicit acoustic echo cancellation (iAEC) framework where a neural network is trained to exploit the additional information from a reference microphone channel to learn to ignore the interfering signal and improve detection performance. We study this framework for the tasks of KWS and DDD on, respectively, an augmented version of Google Speech Commands v2 and a real-world Alexa device dataset. Notably, we show a 56% reduction in false-reject rate for the DDD task during device playback conditions. We also show comparable or superior performance over a strong end-to-end neural echo cancellation baseline for the KWS task with two order of magnitude less computational requirements.

Index Terms— keyword spotting, human computer interaction, front-end processing, speech recognition, acoustic echo cancellation.

1. INTRODUCTION

In the past few years, we have seen a rise in popularity of smart-home devices and virtual assistants. Reliable operation of these systems requires addressing many challenging problems [1] and includes scenarios where user speech can overlap with the device playback audio [2–4]. This phenomenon, sometimes called "barge in" [2, 4], happens as well in human-to-human interactions where one interlocutor begins speaking before another has finished. Such overlap, is especially problematic in human-device interactions, since usually the signal-to-interference ratio (SIR) between the user speech and device playback is low as the device loudspeakers are much closer to the device microphones than the user is. Moreover, even when user speech and device playback do not overlap, if the playback audio consists of speech, e.g. when it is a text-to-speech (TTS)-generated response to an user query or podcast audio, many tasks become arduous. For example, considering custom/multi-KWS [5, 6], without appropriate countermeasures, a model will be prone to pick up also keywords from device TTS playback, especially as TTS models are increasingly realistic or include celebrity-derived custom voices. This can lead to the device "self waking" and continuously interrupting itself as the model, alone, cannot implicitly distinguish between user and device speech and ignore this latter. Such problem also affects automatic speech recognition (ASR) or keyword-less initiated interactions, such as device-directed detection (DDD) [7–10]. One trivial way to mitigate this issue would be disabling the KWS functionality while the device is in playback. Yet, doing so prevents the user to "barge in", making the interaction significantly less natural and intuitive for the user.

Following [2–4], this problem is best formulated in the acoustic echo cancellation (AEC) framework as usually the device playback audio is known. We can thus define the playback *reference signal* $\boldsymbol{r}$ and a mixed signal $\boldsymbol{y} = \Gamma(\boldsymbol{r}) + \boldsymbol{u}$, where $\Gamma(\cdot)$ is a (possibly non-linear) function and $\boldsymbol{u}$ is the *target signal* for the task at hand i.e. the signal which would be captured by the device if there wasn't any playback return signal $\Gamma(\boldsymbol{r})$. This can include instances where there is no user "barging in", i.e. for which $\boldsymbol{u}$ is only background noise, which the classifier should ignore. A common model for $\Gamma(\cdot)$ is to use a linear approximation such that $\boldsymbol{y} = \boldsymbol{r} * \boldsymbol{h} + \boldsymbol{u}$, where $\boldsymbol{h}$ is the impulse response that characterizes the propagation of $\boldsymbol{r}$ and includes effects from the room, speaker, and microphone. We will refer to $\boldsymbol{n} = \Gamma(\boldsymbol{r})$ as the *interferer signal* hereafter. Note that $\boldsymbol{u}$ itself could be the user far-field speech with reverberation and other interferers. In this work however we only focus on robustness against the device playback return signal $\boldsymbol{n}$. We also restrict ourselves to the monaural case, where the mixture signal is single-channel.

AEC techniques can be employed to obtain an estimate of the target $\boldsymbol{u}$ by leveraging the reference signal $\boldsymbol{r}$. These include both classical [2–4, 11–15] and neural-based (nAEC) methods [6, 16–20] which are generally more effective. These latter however require the oracle target signal to be available at training time, which is difficult to obtain directly on real-world data, at least in a scalable way. Thus, nAEC methods rely on synthetic data for training, which is inherently mismatched with respect to real-word data. To counter this mis-

*Work performed while at Amazon Alexa.

match, [6] proposes the use of an additional ASR auxiliary loss obtained from the latent representation of the encoder. Instead [5] leverages ASR in inference and proposes a framework for cancelling the TTS playback interferer $\boldsymbol{n}$ by using the textual source of the reference TTS signal. However, this latter is only applicable to TTS playback and does not generalize to other forms of playback audio such as podcasts or music. In addition, nAEC methods that rely on ASR cannot be used for always-on frontend applications such as KWS since it would be too computationally expensive to perform ASR continuously on resource-constrained edge devices.

Focusing on KWS and DDD tasks, we propose to directly feed the reference signal $\boldsymbol{r}$ as an additional input to the back-end task model together with the mixture signal captured by the device. We show that such access to the reference signal allows the KWS or DDD classifier to disambiguate between the target and the playback return signal and learn to ignore this latter without the need for an AEC or nAEC preprocessing front-end. We call this approach *implicit Acoustic Echo Cancellation* (iAEC). This method allows for considerable computational resources savings as the computational overhead for having a classifier taking also the reference signal in input is extremely low. We show that competitive performance can be obtained with minimal modifications of the network architecture. We study two different strategies for feeding the reference channel to the back-end classifier, one that involves concatenation and another based on latent-representation masking. We found this latter choice especially promising as it allows for the KWS and DDD architecture to be unchanged (no computational overhead) when there is no playback, which is the predominant scenario in deployment.

Importantly, in this paper we consider DDD for always-on, streaming scenarios. Previous DDD works [7–10] performed DDD downstream of a KWS model. These systems are not always running, are more resource intensive, and have higher latency than front-end components. Here instead we consider the use of a DDD model that is run continuously and subsumes the role of a KWS model, allowing for a full keyword-free interaction. Understandably, as it is keyword-free and continuously running, such model is especially affected by the device playback issue and prone to the "self wake" issue.

We perform our experiments using two datasets: for DDD we use a dataset collected from Alexa assistant devices, while for multi-KWS experiments we instead use a purposely developed augmented version of Google speech commands v2 (GSCv2) [21] which simulates a smart-home device in both playback and non-playback scenarios.

Finally, as an additional contribution, we devise a data augmentation strategy that helps avoiding bias in training data, boosts performance and allows to train iAEC and nAEC methods even on data with no device playback conditions.

2. IMPLICIT ACOUSTIC ECHO CANCELLATION

In this work, we focus strictly only on keyword spotting (KWS) and DDD tasks with the goal of devising a computationally efficient strategy to improve the performance of such tasks during device playback, without incurring in a degradation in normal conditions (non-playback).

During playback, a KWS or DDD classifier generally observes only the mixture $\boldsymbol{y} = \Gamma(\boldsymbol{r}) + \boldsymbol{u}$. As said, this presents a challenge when the interferer $\boldsymbol{n} = \Gamma(\boldsymbol{r})$ and target $\boldsymbol{u}$ could lead to an ambiguity for the task at hand (e.g. the interferer $\boldsymbol{n}$ contains a keyword but the target $\boldsymbol{u}$ does not). In such situations, training a classifier on the mixture signal $\boldsymbol{y}$ without access to the reference could bias the model to learn unintended characteristics of the interferer signal. For example, the TTS voice/gender or the fact that the playback interferer has usually more energy than the user speech as the loudspeakers are close to the microphones. These biases could degrade performance severely when the model is deployed. One way to obviate to this is to feed to the KWS or DDD classifier the reference signal along with the mixture signal ones to aid in the classification task. The name iAEC stems from the fact that the model here does not have to explicitly recover the target signal as in AEC methods (an arguably more complex task), but only learn how to use the additional reference input $\boldsymbol{r}$ to ignore the playback return signal $\boldsymbol{n}$ and classify correctly the target signal $\boldsymbol{u}$.

In this framework, the goal is to learn a function $\mathcal{F}\left(\boldsymbol{y}, \boldsymbol{r}, \boldsymbol{\theta}\right)$ parametrized by $\boldsymbol{\theta}$ that models the joint conditional distribution $P(y_\tau|\mathbf{y}, \mathbf{r})$ where y_τ are the labels for the task at hand belonging to the target signal $\boldsymbol{u}$ (e.g. for KWS, keyword or non-keyword). Importantly, this formulation includes non-playback conditions, e.g. for which the reference $\boldsymbol{r}$ and thus the interferer $\boldsymbol{n} = \Gamma(\boldsymbol{r})$ is zero and the mixture signal is simply equal to the target $\boldsymbol{y} = \boldsymbol{u}$. For such instances the problem simply resolves to modeling $P(y_\tau|\mathbf{u})$ as in "classical" KWS or DDD.

2.1. Reference Signal Fusion

2.1.1. Concatenation: iAEC-C

Since we focus in this work on KWS and DDD tasks, where usually features like log Mel-filterbank energies (LFBEs) are employed, we can concatenate the mixture and reference signals feature vectors, respectively $\mathbf{x}_y(k)$ and $\mathbf{x}_r(k)$, and feed them to the classifier. To make notation less cumbersome, we drop the frame index k in the following. This simple method could allow the classifier to exploit the reference channel information.

As depicted in Figure 1, left panel, during playback mode, we apply batch normalization [22] (BN) separately to the reference and mixture branches prior to concatenation. This is done because reference and mixture signal have usually very different gains, as the latter is usually far-field speech. During non-playback conditions, the input is concatenated with the learned bias parameters β of the BN layer. This is equal to concatenating the mean of the post-normalized input features (see left-bottom panel of Figure 1).

2.1.2. *Masking: iAEC-M*

Concatenating the input feature with the learned β parameters from the BN layer is a waste of computing resources in non-playback conditions, which account for most of the deployment time. This problem can be obviated by using the reference to produce a sigmoid mask which is applied to the latent representation of the mixture signal, as illustrated in the right panel of Figure 1.

Compared to iAEC-C, here $\mathcal{F}$ is split into an encoder $\mathcal{E}$ and decoder $\mathcal{D}$. The encoder is shared between reference and mixture branches to save L1 cache memory and produces two latent representations: the reference embedding $\boldsymbol{Z}_r = \mathcal{E}(\boldsymbol{x}_r)$ and the mixture embedding $\boldsymbol{Z}_y = \mathcal{E}(\boldsymbol{x}_y)$.

These latent representations are $\in \mathbb{R}^{T \times D}$, where T is the sequence length and D the embedding size. They are concatenated and used to derive a mask through a linear projection layer $\mathcal{P}$ with weight matrix $\in \mathbb{R}^{2D \times D}$, followed by a sigmoid activation $\sigma(\cdot)$: $\boldsymbol{M} = \sigma(\mathcal{P}([\boldsymbol{Z}_y, \boldsymbol{Z}_r]))$. This mask is then applied to the encoded representation from the mixture branch $\boldsymbol{Z}_\gamma = \boldsymbol{M} \odot \boldsymbol{Z}_y$ via element-wise multiplication. This operation acts as a gating mechanism over $\boldsymbol{Z}_y$. Finally $\boldsymbol{Z}_\gamma$ is fed to $\mathcal{D}$ to obtain the predictions. In non-playback mode the masking mechanism along with the entire reference branch are dropped and $\boldsymbol{Z}_\gamma = \boldsymbol{Z}_y$ directly.

2.2. On-the-fly Data Augmentation

In some application scenarios the reference playback signal $\boldsymbol{r}$ from the device may be not available for training and only the mixture $\boldsymbol{y}$ is available. In edge-applications for example, the playback TTS signal is usually not uploaded to the server-side to save bandwidth. Moreover, regarding smart-home devices, indeed most of the examples available in training can feature limited or no user barge in scenarios as e.g. could be an experimental/not fully supported feature. In these instances one obvious solution is to add artificial device playback via simulation. This may require a complex pipeline involving room, loudspeaker and front-end simulations.

Instead, here, as an additional contribution, we propose a simple but effective on-the-fly data-augmentation strategy to generate $(\boldsymbol{y}, \boldsymbol{r}, \boldsymbol{n})$ triplets by sampling multiple mixture signals $\boldsymbol{y}$ from the training set.

Given a collection of N training examples $\mathbf{x}_l$ and corresponding labels y_l for the task at hand $\{(\mathbf{x}_l, y_l)\}_{1 \dots N}$ we randomly sample two examples $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$. These two examples are arbitrarily assigned the role of target $\boldsymbol{u} = \mathbf{x}_i$ and reference $\boldsymbol{r} = \mathbf{x}_j$ no matter their original labels. We then generate the mixture $\boldsymbol{y}$ by applying a random time-shift to $\mathbf{x}_j$ and mixing it with $\mathbf{x}_i$ at a randomly chosen SIR. The correspond-

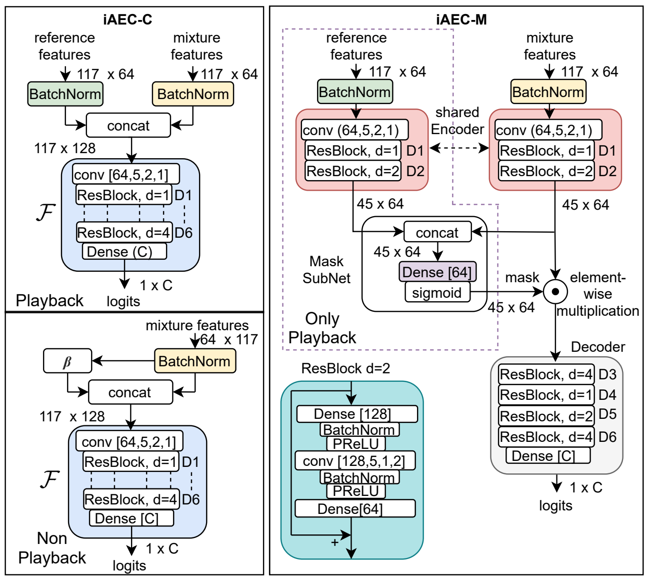

Fig. 1: Schematic of implicit acoustic echo cancellation with concatenation (iAEC-C) and encoder-masking-decoder (iAEC-M) architectures. The LFBE feature extraction part is omitted for simplicity. Neural blocks and input features are described in detail in Section 4.1. Tensor dimensions are represented as *sequenceLength* $\times$ *featureMaps* and we display the values used during training. 1D convolutional blocks (conv) are represented as [*featureMaps*, *kernelSize*, *stride*, *dilation*] while linear layers (Dense) as [*featureMaps*]. Regarding iAEC-M, we show the best configuration (D2) as found in Section 4.2. We also report for convenience the layers used in each ResBlock, which has the same structure as in [24, 25].

ing mixture $\boldsymbol{y}$ is assigned label $y_\tau = y_i$ and the interferer label y_j is ignored. This strategy forces the model to learn to ignore $\mathbf{x}_j$ by using the reference $\boldsymbol{r}$, whatever the original label y_j. As $\mathbf{x}_j$ can contain a keyword from a user, without the reference $\boldsymbol{r}$ it would be impossible for the model to ignore the interferer signal (as it is a legit user keyword) and output the correct prediction relative to the target $\boldsymbol{u}$. Because the target and interferer belong effectively to the same distribution, this data augmentation can also mitigate potential bias in the training data and boosts generalization to new speakers as we show in Section 4.2. Note that it is also possible to leverage unlabeled data when assigning the interferer.

We show that, as far our application is concerned, this naive technique is competitive with simulation via image-method techniques such as gpuRIR [23] and, used as an additional data-augmentation strategy, can improve the results even when the true oracle target, interferer and reference are available in training (see Section 4.3). We show in Section 4.3 that can also boost performance of nAEC models.

3. DATASETS

3.0.1. *Speech Commands Mix*

To study how playback can degrade KWS in a controlled scenario, we extended GSCv2 [21] with TTS and music. We generated two additional versions of the original dataset by mixing TTS from LibriTTS [26] and music from Musan [27], respectively. To generate challenging TTS interferers, we sampled segments from LibriTTS containing GSCv2 keywords. The temporal location of the keywords from LibriTTS was determined using forced alignment. Interferer and reference pairs are generated using gpuRIR [23] by adding artificial reverberation to the original LibriTTS segments. For each room impulse response, we sampled a room size from uniform distribution $U(10, 50)\ m^2$ and T60 reverberation time from $U(0.2, 0.6)$ s. The position of the source is chosen randomly inside the virtual room. To simulate a smart-speaker device, the microphone position is constrained to be in a radius of 5 cm from the source, oriented outward with cardioid polar pattern. Mixture files are obtained by mixing the reverberated interferer signal and original GSCv2 signal with SIR $\sim U(-12, 3)$ dB, which is consistent with what is observed on Alexa devices. An equivalent procedure is followed for mixtures with music. We use the official GSCv2 training, development and test split in our experiments. We made the scripts to generate this dataset available[1].

3.0.2. *Alexa Dataset*

Amazon Alexa assistant provides a "follow-up" mode (FUM) [9] that allows users to interact with the device agent without repeated use of the wakeword. We leverage this data corpus for training and evaluating our DDD model. Ground truth DDD speech annotations are provided for each FUM utterance. The dataset consists of 538k utterances split into train, dev, and test partitions of 465k, 20k, and 53k utterances, respectively. As FUM does not contain any playback data, in our experiments we also used a TTS model to generate synthetic playback signals for the default Alexa voice. In our experiments we also report results when another, unmatched voice (*Matthew*) is used in training. We generated 500k clean TTS utterances and used gpuRIR to add artificial reverberation with the same configuration described in Section 3.0.1. For evaluating our models under TTS playback conditions, we used an Alexa Echo Show 10 device to record both playback-only device responses and instances where 10 speakers were tasked to say commands over the TTS audio. The default TTS Alexa voice was used. This lab collection resulted in a corpus of approximately 2 hours and 45 minutes which was split equally in development and test partitions. Note that both training/dev and test sets can contain background environmental noise in the target signal.

4. EXPERIMENTS

4.1. Architecture and Training Details

For our experiments we employ the Temporal Convolutional Network (TCN) from [25] with a few modifications. Firstly, we use here a smaller network with $X = 3$ residual blocks (ResBlock in Figure 1) and $R = 2$ repeats for a total of 6 blocks. Secondly, we employ an initial 1D convolutional layer with kernel size 5 and stride 2 instead of a bottleneck convolutional layer. Thirdly, BN is used instead of global layer normalization [24] and depth-wise convolutions in ResBlocks have kernel size of 5. A final linear layer with output size C is used to derive the class logits. A sigmoid activation function is used for DDD ($C = 1$) while softmax is used for multi-KWS experiments ($C = 35$ posteriors, corresponding to the number of keywords in GSCv2, both original and our augmented version). The total number of parameters is 131k. We use 64-dimensional LFBEs as input features extracted with a Short-Time Fourier Transform (STFT) 25 ms window and 10 ms stride. The model is trained using cross-entropy loss on segments of 117 frames, which is the receptive field of the model. If the input length is less than 117 frames, zero padding is employed. During testing, we employ max pooling if the input feature sequence is longer than the receptive field producing a single prediction for the whole sequence. For regularization, we apply SpecAugment [28] independently on both reference and input mixture LFBE features after the initial BN layers. We use the Adam algorithm [29] for optimization and tune the learning rate, weight decay and SpecAugment hyper-parameters for each experiment using the validation split. Each model is trained for a maximum of 200 epochs with 10 epochs early stopping. We use a batch size of 256 and 1200 for the multi-KWS and DDD experiments, respectively. All models are trained on both playback (where reference is available) and non-playback conditions.

Since both KWS and DDD models employ LFBEs features in input, in our experiments we perform the proposed on-the-fly data augmentation strategy described in Section 2.2 in the STFT domain, prior to taking the magnitude and apply the Mel filterbank transform. This leads to a minor training speed-up. The time-shift is applied on STFT frames and is sampled from $U(15, 20)$ frames. The interferer and target signals are mixed such that the SIR falls in $U(-20, 3)$dB. Note that such rather extreme frame-wise shifting is necessary to simulate the non-deterministic delay in the device playback and input audio pipeline, mainly due to input/output audio software buffers. This delay leads to substantial misalignment between the reference $\boldsymbol{r}$ and corresponding interferer $\boldsymbol{n} = \Gamma(\boldsymbol{r})$ and must be accounted for during training so the model learns to compensate for it.

4.2. Device-Directed Speech Detection

In this section we present and discuss our DDD experiments on the real world dataset described in Section 3.0.2 focusing on TTS playback conditions. We use false reject rate (FRR) and false accept rate (FAR) as our metrics and report FRR at two fixed FAR values (non-playback and playback). The FAR values are chosen on the development set according to customer feedback and are redacted in this document due to privacy reasons. As our goal is to obtain a practical on-device streaming DDD classifier, we also report the floating point

[1]https://github.com/popcornell/SpeechCommandsMix

operations (FLOPs) per output prediction in both playback and non-playback conditions.

In Table 1 (upper panel) we report the results for adding simulated interferer signals while training a standard TCN DDD classifier (Baseline) with the architecture explained in Section 4.1. Here, we compare the strategies of using a test and dev set matched (default Alexa voice) versus unmatched (Matthew voice) TTS model during training. As expected, adding simulated playback with a matched TTS model (+ Alexa TTS) significantly improves performance especially in playback. On the contrary if an unmatched TTS model is used in training, very marginal improvement is observed in playback while in non-playback conditions FRR degrade significantly. This suggests that the model is learning to ignore the TTS playback mainly based on the "identity" and, to a less extent, perceived gender of the TTS speech. The FRR in non-playback increases for the second model because his perceived gender is male, and the test set is composed mainly by male speakers (while the Alexa TTS voice perceived gender is female). This biases the model to be more prone to consider male speakers as TTS and reject them.

In the second panel, we study the effect of extending the classifier by simply concatenating the reference channel features at the first layer, after BN, as explained in Section 2.1.1. This model requires slightly more FLOPs than the standard Baseline TCN classifier. As our training dataset lacks playback conditions (see Section 3.0.2) also here we resort to simulation using both matched and unmatched TTS voices. However we also study the effect of the data augmentation strategy outlined in Section 2.2 which can be leveraged now since the model (iAEC-C) has access to the reference.

The proposed data augmentation strategy alone (iAEC-C augm) is able to improve FRR in both conditions despite the model is trained with no TTS data but only with legit user queries used both for targets and playback roles. Adding simulated matched and, to a less extent, even unmatched TTS interferer/reference data further improves the performance in both conditions. In the Matthew TTS case, the proposed method helps fighting potential biases in the training data and prevent overfitting one particular voice/gender. On the other hand if the iAEC-C model is trained only with matched TTS simulated data and no data augmentation strategy (- *augm* + Alexa TTS) we observe a degradation in performance. This hints that the model with reference access is more likely to overfit the simulated interferer acoustic characteristics in the training set and not generalize well to real-world environments despite our simulation efforts. More complex simulation pipelines may be able to mitigate this issue but have their drawbacks (e.g. lots of hand-tuning). As smart-home assistants offer more voice options, the proposed data augmentation allows to avoid retraining the DDD classifier for each new TTS voice and reduces the risk of rejecting speakers whose voices are similar to the TTS model.

The top panel of Table 2 investigates the effect of concatenating at deeper residual blocks for iAEC-C from the 1st (D1, same as in Table 1) to 3rd (D3) blocks (see Figure 1). All models are trained using the data-augmentation (*augm*) strategy outlined in Section 2 with no simulated playback TTS data. Performance improves with deeper layers up to D2, as the encoder receptive field (FOV) surpasses the maximum offset between reference and interferer signals observed on real-world collected audio, around 200 ms, as said, mainly due to the output and input software audio buffers. On the other hand, also computation increases with the depth at which concatenation is performed. We can see that concatenation at D2 provides the best trade-off between playback FRR, non-playback FRR, and FLOPs.

Table 1: FRR at fixed FAR (redacted) for non-playback and playback (TTS playing) conditions. We study different training strategies using the default Alexa voice and an alternative TTS voice named Matthew. Intra-dataset mixing (intramix) refers to the on-the-fly mixing strategy outlined in Section 2.

Model	**Non-Playback**		**Playback**	
	FRR@FAR	FLOPs	FRR@FAR	FLOPs
Baseline	0.189	242k	0.348	242k
+ Alexa TTS	0.187	–	0.241	–
+ Matthew TTS	0.202	–	0.339	–
iAEC-C (augm)	0.188	283k	0.258	283k
+ Alexa TTS	**0.185**	–	**0.227**	–
+ Matthew TTS	0.187	–	0.256	–
- augm + Alexa TTS	0.193	–	0.299	–

Table 2: FRR at fixed FAR (redacted) using iAEC-M and iAEC-C with concatenation at different layers in the TCN model.

Model		**Non-Playback**		**Playback**	
	FOV	FRR@FAR	FLOPs	FRR@FAR	FLOPs
iAEC-C (input)	5	0.188	283k	0.258	283k
iAEC-C (D1)	12	0.183	336k	0.178	336k
iAEC-C (D2)	28	0.184	369k	0.165	369k
iAEC-C (D3)	60	0.183	402k	0.168	402k
iAEC-M (D2)	28	**0.181**	242k	**0.150**	367k

The bottom panel of Table 2 presents the mask-based approach described in Section 2.1.2. Masking is applied at the second residual block (D2) as in iAEC-C and shows a further performance improvement on both playback and non-playback conditions. Regarding computational efficiency at D2, iAEC-M requires slightly more FLOPs in playback mode than iAEC-C, but significantly less FLOPs in non-playback mode, as the reference branch is dropped and the architecture becomes equal to a standard classifier. Since this model is ran continuously, playback conditions account for a very small fraction of total inference time and thus iAEC-M leads to the best performance/FLOPs trade-off.

4.3. Multi Keyword Spotting

In Table 3 we report our results on the augmented GSCv2 dataset described in Sec 3.0.1. We report keyword-detection accuracy over the 35 possible GSCv2 keywords for the 3 different test sets in our augmented version: original GSCv2 (*Non-Playback*), mixed with music (*Playback Music*) and with TTS (*Playback TTS*).

As with the DDD experiments, we use a standard TCN classifier without access to the reference as our *Baseline*. We also consider a joint model (*+nAEC*) comprised of the state-of-the-art neural AEC model from [6], designed for edge-devices applications, and the *Baseline* TCN classifier: the output of the nAEC is directly fed to the classifier in cascade. Moreover we compare with an STFT-based normalized least mean squares (NLMS) acoustic echo cancellation (AEC) algorithm with 32 taps, step size $\mu = 0.5$, 512 and 128 STFT window and hop with square-root Hann window. Pyroomacoustics [30] was used for the implementation.

For this system we use a weighted loss consisting of a spectral loss term as in [6] and a KWS loss term instead of an ASR on as in [6]. The two cascaded models nAEC and TCN are then jointly trained. As another baseline, we include the performance for MatchBoxNet-3x2x64 [31] (MatchBN) using the official implementation from NeMo toolkit [32]. We also report results for three different training strategies for the models with access to the reference (thus including [6]). In *orcl* we train the model with access to the oracle reference, interferer and target signals, e.g. the nAEC model is trained to estimate the target and is given in input the reference corresponding to the interferer together with the mixture signal. This is a best-case scenario, possible because this dataset is fully synthetic. Here there is no mismatch between training and test data, a rather ideal condition which e.g. will prevent the degradation observed in DDD experiments in Table 1 last row due to mismatched simulated training and real-world interferer acoustic conditions. The *augm* denotes the augmentation strategy described in Section 2.2 where, in training, we generate fake playback mixture signals by mixing original GSCv2 examples. Each example is randomly given the role of interferer/reference or target, without adding any simulated playback TTS or music. Finally the strategy denoted *both* denotes a combination of these two: in training one example is either generated on-the-fly with *augm* or comes from *orcl* with 50% probability.

As expected, we can observe that for both *Baseline* and *MatchBN* playback performance degrades significantly, especially during TTS playback conditions, which can confuse the model. Regarding the models with access to the reference (Baseline +nAEC, iAEC-C and iAEC-M), we can see that, in the best case scenario of matched training and testing (*orcl*), all models significantly improve performance over the Baseline classifier especially in playback conditions. The improvement in non-playback is due to the fact that the MatchBN and Baseline models are prone to reject a valid keyword as TTS playback, as they have no access to the reference. If only the *augm* strategy is used in training the performance degrades compared to the ideal *orcl* data but still affords improvement over models with no reference access. On the other hand, as explained, in many real-world applications the reference signals can be difficult to obtain or may be biased (e.g. all examples with playback belongs to one TTS model). Combining both strategies yields the best performance overall for all models (including nAEC) even surpassing *orcl*. The NLMS AEC is very effective with the music interferer but struggles with TTS. On the other hand it is extremely light computationally and thus future work could explore combinations of this approach and the proposed one.

Overall, our proposed iAEC approaches perform competitively with *+nAEC* but uses two order of magnitude fewer FLOPs for each prediction. This makes the proposed methods more suitable for always-on low-resource applications.

Table 3: Accuracy and FLOPs on GSCv2 for different models and training strategies (see Sec 4.3). Metrics are reported both for original data (*Non-Playback*) and our simulated playback corpus, separately for *TTS* and *Music* playback conditions.

Model		**Non-Playback**		**Playback**		
		Acc %	FLOPs	Acc % (Music)	Acc % (TTS)	FLOPs
MatchBN [31]		94.47	185k	61.84	36.78	185k
Baseline		93.35	242k	75.01	61.71	242k
+ NLMS AEC		93.77	242k	78.95	63.13	243k
+nAEC	orcl	94.06	242k	82.87	82.75	15M
	augm	93.82	-	75.04	72.21	-
	both	94.46	-	83.55	**83.81**	-
iAEC-C	orcl	94.52	283k	82.60	80.79	283k
	augm	94.56	-	77.91	77.52	-
	both	94.74	-	83.54	82.93	-
iAEC-M	orcl	94.67	242k	83.87	82.47	367k
	augm	94.49	-	78.21	77.01	-
	both	**94.97**	-	**84.22**	83.79	-

5. CONCLUSIONS & FUTURE WORK

In this work, we address the problem of boosting KWS and DDD classifiers performance on edge-devices during device playback without degrading non-playback performance. In such scenarios, the playback signal is usually known. We propose a simple and computationally efficient implicit acoustic echo cancellation (iAEC) framework where a classifier leverages the known playback signal (reference signal) to ignore the return “echoed” playback signal (interferer signal) captured by the device microphones together with the user speech.

We explored two strategies for feeding the reference signal to our models and found the use of a latent-space masking approach particularly suited for our KWS and DDD tasks, as it brought significant performance improvements in device playback conditions. On the KWS task the proposed method obtains comparable performance with a state-of-the-art neural AEC method but with much less computational requirements.

As an additional contribution, we devised an effective data augmentation strategy that is able to further boost performance of nAEC and iAEC models, allows to train such models on examples with no playback interferer and helps reducing bias in the training data.

Future work could explore other tasks such as ASR and scenarios beyond device playback where oracle double-talk detection is not available.

6. REFERENCES

[1] R. Haeb-Umbach, S. Watanabe, T. Nakatani, M. Bacchiani, B. Hoffmeister, M. L. Seltzer, H. Zen, and M. Souden, "Speech processing for digital home assistants: Combining signal processing with deep-learning techniques," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 111–124, 2019.

[2] R. Takeda, K. Nakadai, T. Takahashi, K. Komatani, T. Ogata, and H. G Okuno, "Ica-based efficient blind dereverberation and echo cancellation method for barge-in-able robot audition," in *Proc. ICASSP*, 2009, pp. 3677–3680.

[3] R. Takeda, K. Nakadai, T. Takahashi, K. Komatani, T. Ogata, and H. G. Okuno, "Efficient blind dereverberation and echo cancellation based on independent component analysis for actual acoustic signals," *Neural computation*, vol. 24, no. 1, pp. 234–272, 2012.

[4] I. Nishimuta, K. Yoshii, K. Itoyama, and H. G Okuno, "Development of a robot quizmaster with auditory functions for speech-based multiparty interaction," in *SICE International Symposium on System Integration*, 2014, pp. 328–333.

[5] S. Ding, Y. Jia, K. Hu, and Q. Wang, "Textual echo cancellation," *arXiv preprint arXiv:2008.06006*, 2020.

[6] N. Howard, A. Park, T. Z. Shabestary, A. Gruenstein, and R. Prabhavalkar, "A neural acoustic echo canceller optimized using an automatic speech recognizer and large scale synthetic data," in *Proc. ICASSP*, 2021, pp. 7128–7132.

[7] S. H. Mallidi, R. Maas, K. Goehner, A. Rastrow, S. Matsoukas, and B. Hoffmeister, "Device-directed utterance detection," *Proc. Interspeech*, pp. 1225–1228, 2018.

[8] C. Huang, R. Maas, S. H. Mallidi, and B. Hoffmeister, "A study for improving device-directed speech detection toward frictionless human-machine interaction," *Proc. Interspeech*, pp. 3342–3346, 2019.

[9] K. Gillespie, I. C. Konstantakopoulos, X. Guo, V. T. Vasudevan, and A. Sethy, "Improving device directedness classification of utterances with semantic lexical features," in *Proc. ICASSP*, 2020, pp. 7859–7863.

[10] A. Norouzian, B. Mazoure, D. Connolly, and D. Willett, "Exploring attention mechanism for acoustic-based classification of speech utterances into system-directed and non-system-directed," in *Proc. ICASSP*, 2019, pp. 7310–7314.

[11] J. Benesty, T. Gänsler, D. R. Morgan, M. M. Sondhi, S. L. Gay, et al., *Advances in network and acoustic echo cancellation*, Springer, 2001.

[12] E. Hänsler and G. Schmidt, *Acoustic echo and noise control: a practical approach*, vol. 40, Wiley-Interscience, Hoboken, 2005.

[13] D. A. Bendersky, J. W. Stokes, and H. S. Malvar, "Nonlinear residual acoustic echo suppression for high levels of harmonic distortion," in *Proc. ICASSP*, 2008, pp. 261–264.

[14] A. Schwarz, C. Hofmann, and W. Kellermann, "Spectral feature-based nonlinear residual echo suppression," in *Proc. ICASSP*, 2013, pp. 1–4.

[15] B. Panda, A. Kar, and M. Chandra, "Non-linear adaptive echo supression algorithms: A technical survey," in *Proc. ICASSP*, 2014, pp. 076–080.

[16] H. Zhang and D. Wang, "Deep learning for acoustic echo cancellation in noisy and double-talk scenarios," *Training*, vol. 161, no. 2, pp. 322, 2018.

[17] Q. Lei, H. Chen, J. Hou, L. Chen, and L. Dai, "Deep neural network based regression approach for acoustic echo cancellation," in *Proc. ICMSSP*, 2019, pp. 94–98.

[18] A. Fazel, M. El-Khamy, and J. Lee, "Deep multitask acoustic echo cancellation.," in *Proc. Interspeech*, 2019, pp. 4250–4254.

[19] H. Zhang, K. Tan, and D. Wang, "Deep learning for joint acoustic echo and noise cancellation with nonlinear distortions.," in *Proc. Interspeech*, 2019, pp. 4255–4259.

[20] A. Fazel, M. El-Khamy, and J. Lee, "Cad-aec: Context-aware deep acoustic echo cancellation," in *Proc. ICASSP*, 2020, pp. 6919–6923.

[21] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, 2018.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

[23] D. Diaz-Guerra, A. Miguel, and J. R. Beltran, "gpuRIR: A python library for room impulse response simulation with gpu acceleration," *Multimedia Tools and Applications*, vol. 80, no. 4, pp. 5653–5671, 2021.

[24] Y. Luo and N. Mesgarani, "Conv-TasNet: Surpassing Ideal Time–Frequency Magnitude Masking for Speech Separation," vol. 27, no. 8, pp. 1256–1266, Aug. 2019.

[25] S. Cornell, M. Omologo, S. Squartini, and E. Vincent, "Detecting and counting overlapping speakers in distant speech scenarios," in *Proc. Interspeech*, 2020.

[26] H. Zen, R. Clark, R. J. Weiss, V. Dang, Y. Jia, Y. Wu, Y. Zhang, and Z. Chen, "Libritts: A corpus derived from librispeech for text-to-speech," in *Proc. Interspeech*, 2019.

[27] D. Snyder, G. Chen, and D. Povey, "Musan: A music, speech, and noise corpus," *arXiv e-prints*, pp. arXiv–1510, 2015.

[28] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple augmentation method for automatic speech recognition," in *Proc. Interspeech*, 2019.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić, "Pyroomacoustics: A python package for audio room simulation and array processing algorithms," in *Proc. ICASSP*. IEEE, 2018, pp. 351–355.

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

[32] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, et al., "Nemo: a toolkit for building ai applications using neural modules," *arXiv preprint arXiv:1909.09577*, 2019.