Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 657–666

DOI: 10.15439/2023F8594
ISSN 2300-5963 ACSIS, Vol. 35

# Open Vocabulary Keyword Spotting with Small-Footprint ASR-based Architecture and Language Models

Mikołaj Pudo
0000-0002-0776-4703
Samsung R&D Institute Poland, Krakow, Poland
Warsaw University of Technology, Warsaw, Poland
Email: m.pudo@samsung.com

Mateusz Wosik
0009-0002-9530-8931
Samsung R&D Institute Poland, Krakow, Poland
Email: m.wosik@samsung.com

Artur Janicki
0000-0002-9937-4402
Warsaw University of Technology, Warsaw, Poland
Email: artur.janicki@pw.edu.pl

***Abstract*—We present the results of experiments on minimizing the model size for the text-based Open Vocabulary Keyword Spotting task. The main goal is to perform inference on devices with limited computing power, such as mobile phones. Our solution is based on the acoustic model architecture adopted from the automatic speech recognition task. We extend the acoustic model with a simple yet powerful language model, which improves recognition results without impacting latency and memory footprint. We also present a method to improve the recognition rate of rare keywords based on the recordings generated by a text-to-speech system. Evaluations using a public testset prove that our solution can achieve a true positive rate in the range of 73%–86%, with a false positive rate below 24%. The model size is only 3.2 MB, and the real-time factor measured on contemporary mobile phones is 0.05.**

## I. Introduction

MACHINE learning techniques are being developed nowadays in two distinct directions. In some applications, model sizes are constantly growing to support the increasing number of domains. This is especially visible in the case of large language models (LLM), in which the number of trainable parameters reaches hundreds of billions. Those models require tremendous amounts of computing power for inference. However, there is also a trend pushing the boundaries in the opposite direction by minimizing the model sizes. In this case, the models are designed for very specific tasks and they are most commonly deployed on devices with a limited amount of computing power, such as mobile phones or home appliances.

A good example of such a specific task is keyword spotting in an audio stream. This task aims to detect all occurrences of the given keywords in audio data provided either in streaming or non-streaming mode. Systems that solve keyword-spotting tasks are usually deployed on users' devices. Therefore, they need to have low latency and a small memory footprint. In the most basic case, the models are fitted to support detecting only a fixed number of keywords (e.g., wake words in contemporary voice assistants such as "OK Google", "Alexa", "Hey Siri" or "Hi Bixby"). Such models can be minimized well, even to sizes below 100 kB. However, users of voice assistants often request the possibility to customize the keywords, which introduces an open-vocabulary Keyword Spotting (KWS) problem. In this case, the model needs to be much larger to support the recognition of potentially arbitrary keywords.

In this paper, we present our solution to the KWS task for the non-streaming mode. It is based on the acoustic model (AM) architecture used in automatic speech recognition (ASR). However, to fit the entire system (model and engine) on the mobile device, we strongly reduced neural network layer sizes and applied post-training weights quantization. As expected, the baseline model performance was far from satisfactory. Therefore, we applied hypothesis re-scoring with a simple language model (LM). We also explored the idea of using recordings generated by a text-to-speech (TTS) model to improve performance on rare keywords not known during AM training. It should be noted that both improvements can be used independently of each other and can be applied to any type of AM.

The rest of this paper is organized as follows. In Section II, we discuss previous solutions to the KWS problem. In Section III, we present different parts of the model architecture: AM in Section III-A; LM together with the algorithm of its construction in Section III-B; keyword classifier, which makes the final decision is described in Section III-C; and extension of this module to multiple keywords is explained in Section III-D. All the variants of our solutions discussed in this paper were evaluated. Results of those experiments are presented in Section IV. Finally, we summarize this paper in Section V and provide a selection of possible future research directions in Section VI.

## II. Related work

KWS can be split into two types: query-by-text (QbyT) and query-by-example (QbyE). In QbyT the keyword is provided by text, while in QbyE one or more "enrollment" audio recordings are provided during the initialization phase. Both types of KWS were considered before.

A thorough review of QbyT solutions can be found in [1]. It should be noted that currently, the most popular testset used to evaluate such solutions is a subset of Google Speech Commands (GSC) [2]. It is a public dataset developed for training and evaluating models designed for simple command recognition. It is also used for the KWS task. GSC contains a small number of keywords, hence it is more suited for classification problems with a fixed number of classes rather than open-vocabulary tasks.

Solutions designed for open-vocabulary QbyT evolved similarly to the ASR models. There was a long phase of solutions based on the hidden Markov model (HMM) – Gaussian mixture model (GMM) architecture [3], [4]. Later the GMM component was replaced by deep neural networks (DNN) [5], [6]. Finally, with the advent of sequence-to-sequence architectures to speech processing, ideas such as connectionist temporal classification (CTC) [7] and attention mechanism [8] became standard solutions in KWS as well.

Present-day solutions to open-vocabulary KWS can be based simply on CTC. In [9], the model contains three long short-term memory (LSTM) layers and the output is at the character level. The keyword is detected once the negative log posterior is below a predefined threshold. LSTM-CTC architecture is also used in [10]. However, in this case, the model operates at the phonetic level. The keyword is represented by one or more phone sequences. During the inference phase, those variants are compared with the hypothesis using minimum edit distance. The decision threshold is estimated for each keyword separately based on the training data and the lexicon.

The connection between KWS and ASR can be also limited to the training phase. In [11], the model is trained using CTC in a multi-task approach with ASR and KWS outputs. During inference, only the KWS output is used. Such an approach is intended to improve the model's ability to generalize and improve the performance in acoustically challenging conditions. The solution presented in [12] is based on an audio encoder network and a convolutional classifier. The encoder network is trained using the ASR task. The classifier network uses filters computed by a keyword encoder. The keyword encoder is a bidirectional LSTM (BiLSTM) layer processing the text keyword provided by the user.

Some solutions are based on the attention mechanism. An ASR model composed of five LSTM layers is used in [13]. The model is trained with CTC loss but has an additional keyword encoder and attention network which is used to direct the prediction network towards the keyword of interest. One of the models presented in this paper employs a phoneme level n-gram LM, which improves the model's performance. An attention-based model is presented in [14]. However, this is one of many solutions with a fixed-size output layer, hence supporting new keywords requires retraining the model.

Another approach to supporting open vocabulary is the on-the-fly adaptation of the model during the initialization phase. In [15] an embedding model is pre-trained on a large number of classes. A classification layer for specific keywords is added on top of the embedding model and adapted using only a handful of samples. Such classification layers can be independent of each other and use the same embedding model, since in the adaptation phase only the last layer is modified. A similar solution based on a few-shot transfer learning is described in [16]. It should be noted that usually KWS solutions are deployed on devices with limited resources, hence performing any type of model adaptation might be troublesome.

Many QbyE solutions are also based on the concepts used in the ASR. In [17] the ASR model with CTC is applied to enrollment phase recordings. N-best phonetic level keyword labels are stored together with their log probabilities in the keyword model. During the inference phase, each audio is processed with a similar ASR model. For each keyword from the keyword model log probability is computed and added to the final score. The keyword is detected if the score is above a certain pre-determined threshold. Similarly in [18] a small-footprint ASR model based on CTC is employed. In the enrollment phase, phonetic level posteriorgrams obtained from the model are used to build a finite-state transducer graph (FST) that models the keywords. In the inference phase, the audio is processed with the ASR model, and the output is scored using the keyword model FST. Finally, the score is compared with the threshold, which is chosen automatically based on the enrollment recordings and negative samples generated by rearranging each enrollment waveform. Since both of those solutions operate on the phonetic level rather than directly on the acoustic level, they can be treated as converting the QbyE task to QbyT.

QbyE can be also approached on the acoustic level. In [19], [20], [21], [22] audio embeddings are computed for both enrollment and inference phase recordings. Distance between those vectors is computed using different metrics and compared to a predefined threshold.

## III. Model architecture

The main assumption in the KWS task is that the keyword might be any phrase, most likely not known during model training. This means the model architecture cannot be based on a classifier with a fixed-size softmax-type output layer. A more elaborate solution is necessary for this problem. Fig. 1 provides a general overview of our solution. We decided to adopt the AM architecture developed for the ASR task. To gain high accuracy, AMs usually contain hundreds of millions of trainable parameters. However, since KWS is simpler than speech recognition, we decided to leverage knowledge distillation to minimize the model size, but still keep the capability of dealing with large or open vocabulary, which is the base of

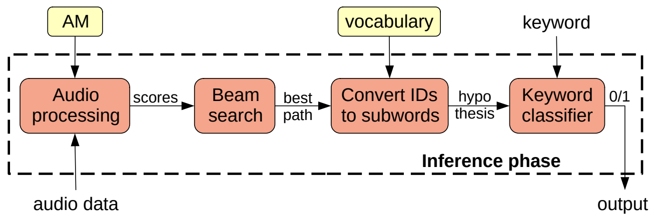

Fig. 1. Overview of the baseline solution.

ASR. Our solution employs AM in a standard way to generate frame level subwords. This is followed by a beam search to create the best path, which is converted to the final hypothesis using model vocabulary. The last step consists of keyword classification, which compares the hypothesis with the given keyword to make a final binary decision: whether a keyword is present in the recording or not.

### *A. Acoustic model architecture and training*

The AM used in our solution is based on monotonic chunkwise attention (MoChA) [23]. It is a sequence-to-sequence model split into encoder and decoder parts, both composed of recurrent neural layers (RNN). The encoder computes embedding vectors for each frame in the audio stream. The attention-based decoder combines and transforms those embeddings into a series of subwords that will constitute the hypothesis. The MoChA used in our decoder is a modification of soft attention, designed to address the issue of real-time (online) processing. It consists of two attention layers. The first layer uses hard monotonic attention for each frame to determine whether the second layer needs to be run. The second layer uses soft attention over a small sequence (“chunk”) of frame embeddings to compute the context vector. Each chunk comprises the current frame and several frames preceding it. The chunk length is a model hyperparameter.

To compress the model, we use a knowledge distillation approach, in which a small student model is trained to mimic a large teacher model [24]. Both models are based on MoChA architecture but differed in the sizes of the selected layers.

The teacher model encoder consists of six BiLSTM layers with 512 units for each direction with 0.3 dropout. To reduce the time domain size, max-pooling layers are used after each of the three initial BiLSTM layers. The decoder consists of one unidirectional LSTM layer with 1000 units, an embedding layer of size 621, and a readout layer of size 1000. The decoder also contains two attention layers of size 512 each. Chunk size two is used for chunkwise attention. The teacher model has 50.1 million trainable parameters (34.2 million in the encoder and 15.9 million in the decoder). It was trained jointly with CTC loss and categorical cross-entropy loss [25]. The CTC was used with the encoder output to encourage the model to learn monotonic alignments. We used 0.1 label smoothing of the output softmax distribution.

The student model has the same architecture as the teacher model but with reduced layer sizes. Each of the BiLSTM encoder layers has 124 units for each direction with 0.3 dropout. The decoder comprises an LSTM layer with 256 units, an embedding layer of size 156, a readout layer of size 256, and two attention layers of size 124 each. The student model includes a total of 3.1 million trainable parameters (2.0 million in the encoder and 1.1 million in the decoder).

The input to both models consists of 40-dimensional power-mels computed over a period of $25\,\mathrm{ms}$ with a $10\,\mathrm{ms}$ step. During training and inference, we applied cepstral mean and variance normalization, which were computed over all training samples. The model's output consists of 500 subwords obtained using the method described in [26]. It uses the adaptation of byte pair encoding (BPE) to word segmentation to generate a compact symbol vocabulary of variable-length subword units. The vocabulary was generated from all the transcriptions contained in the training and testing sets. 500 was chosen as the vocabulary length since this size keeps the output model layer small and allows for more accurate recognition of rare or out-of-vocabulary words. Four special subwords were added to the vocabulary: `<s>` (beginning of a sentence), `</s>` (end of a sentence), `<unk>` (non-speech events or characters not included in the Latin alphabet) and `<blank>` (required for CTC training).

The teacher model was trained for 23 epochs with a learning rate equal to $1 \times 10^{-4}$ in the first epoch. The learning rate was reduced by a factor of $0.95$ after every $10$ consecutive validation steps without change. Validation was done every $5000$ steps. During training, we added randomly selected room impulse response (RIR) and mixed the data with a randomly selected noise signal with a ratio between $-2$ and $12\,\mathrm{dB}$. RIR dataset contains simulations of distances from one to five meters and reverberation time between $0.2\,\mathrm{s}$–$0.9\,\mathrm{s}$. Noise data consisted of both internal noise dataset and AudioSet [27]. The internal noise data contained audio from various environments and is similar to MUSAN [28]. Moreover, the features were augmented with SpecAugment [29] by masking one frequency block of size eight and one time-domain block of size 50.

The student's total loss was the weighted sum of distillation loss and categorical cross-entropy loss with weights of $0.4$ and $0.6$, respectively. During knowledge distillation, we used temperature two to make teacher predictions softer. The student model was trained for 22 epochs with a learning rate equal to $4 \times 10^{-4}$ in the first epoch. The same learning rate scheduler and parameters were utilized during student model training as for the teacher model. We observed accuracy degradation while using SpecAugment during student model training; therefore, we skipped this type of augmentation.

Both models were trained on generic ASR datasets. We used LibriSpeech [30] (all training splits, $960\,\mathrm{h}$), $1779\,\mathrm{h}$ of English Mozilla Common Voice [31] (version 7.0, excluding sentences selected for testing). Additionally, we used $4\,\mathrm{h}$ of audio data not containing speech (silence or quiet noise). The sampling rate of all audio data was $16\,\mathrm{kHz}$. During training, we used greedy decoding, while in the inference phase, we applied the beam search algorithm with a beam size equal to four.

To further minimize student model size, we applied post-training 8-bit quantization. This step reduced the model size

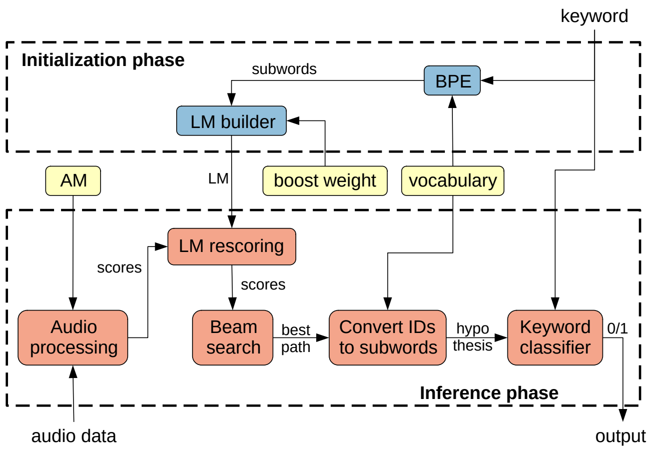

Fig. 2. Overview of the solution with static LM.

from 13 MB to 3.2 MB. This model achieved the following word error rates (WER): 16.0 on LibriSpeech test-clean and 30.9 on LibriSpeech test-other.

## *B. Language model architecture and initialization*

LM can be used to modify scores generated by AM. This process is known as re-scoring. We decided to use a very simple 1-gram LM, where the model is a vector of the same length as the AM output layer size. Such a type of LM introduces only a minor additional memory footprint and a small increase in latency. With this kind of LM, re-scoring consists of element-wise multiplication of the scores returned by the AM and the LM vector. It is the initialization of weights that is the key to a LM of this type. This step should be performed only once for each novel keyword; hence it does not influence latency during the inference phase.

We decided to use a very simple initialization method which we called Static LM. In this method, LM weights are initialized only with two values: one and *boost* weight which is treated as a model hyper-parameter. A general overview of the Static LM method is shown in Fig. 2. The BPE algorithm is applied to the keyword with the same vocabulary as used to convert the AM scores to obtain the hypothesis. Subwords included in the keyword are assigned a *boost* weight, and all the remaining subwords are assigned 1. Note that setting *boost* weight to 1 will not change AM scores and setting *boost* weight to values smaller than 1 will decrease the probability of recognizing the keyword.

## *C. Keyword classifier*

Generating an ASR-based hypothesis is only the first step in the KWS solution. Based on this hypothesis, it is necessary to decide whether the recording contains the required keyword or not. The pseudocode of the procedure we employed for this purpose is presented in Algorithm 1. Note that the recording which is processed by the AM might contain more speech data than just the keyword. To remedy this issue, we calculate the keyword length as the number of words and compare it with all the subsequences of the hypothesis of the same word length. We calculate the character level normalized Levenshtein distance between each such subsequence and the keyword. If the distance is smaller than a predefined threshold, the true value is returned by the system (keyword detected) and the false value is returned otherwise (keyword not detected).

**Algorithm 1** Keyword classifier algorithm

```
Input: $keyword$ – custom keyword
Input: $hyp$ – hypothesis returned by AM
Input: $t$ – recognition threshold
1: $l \leftarrow len(keyword)$ {number of words in keyword}
2: for $s \in \{sub : sub$ is substring of $hyp \wedge len(sub) = l\}$ do
3:     if $dist(keyword, s) \leq t$ then
4:         return $true$
5:     end if
6: end for
7: return $false$
```

## *D. Multi-keyword classifier*

In the generic text-based KWS task, the keyword is provided by the text. However, often additional audio data can be leveraged to improve recognition rates. This can be done for example by requesting the user to provide a spoken version of the keyword. Since such an approach requires additional action from the user, we decided to pursue an automatic solution. An overview of this method is presented in Fig. 3. We employ the TTS system to generate synthetic recordings representing the spoken version of the keyword. The number of those recordings depends on the TTS solution and can also be set as a parameter of the system. Each of those recordings is processed by the AM, followed by the beam search algorithm to generate a hypothesis. The original keyword is appended to the hypothesis list and duplicates are removed. This list is treated as containing additional variants of the original keyword and is later used during inference by the keyword classifier.

Algorithm 2 presents the multi-keyword classifier pseudocode. It is the extended version of the keyword classifier described in section III-C. Once more substrings of the hypothesis are selected for comparison, but this time they are compared with each keyword from the list prepared during the initialization phase. As previously, normalized character level Levenshtein distance and a predefined threshold are used to make the final decision.

The main idea behind the multi-keyword classifier is to improve the recognition rate on keywords that are very distinct from the phrases presented to the AM during training (eg. named entities or other non-standard phrases). In such cases, the AM most likely would return an incorrect hypothesis. However, provided those errors are similar across different samples of the same keyword, adding them to the classifier should increase the true positive rate (TPR). Hence this idea can be described as adding additional pronunciation variants

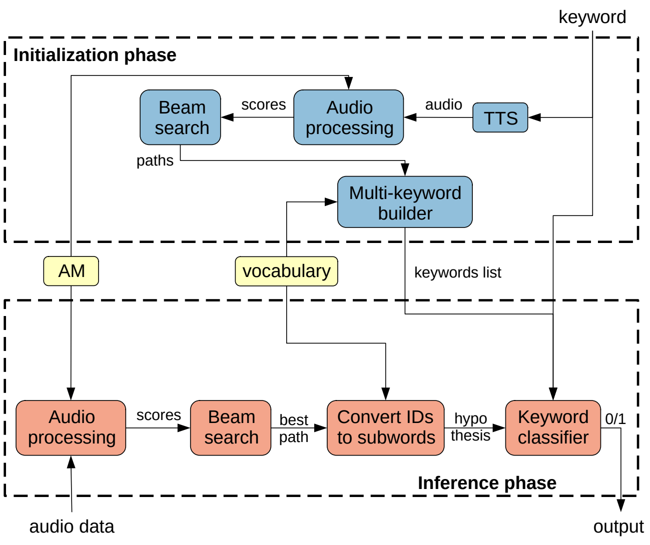

Fig. 3. Overview of the multi-keyword solution.

**Algorithm 2** Multi-keyword classifier algorithm

```
Input: $keywords$ – list of custom keywords
Input: $hyp$ – hypothesis returned by AM
Input: $t$ – recognition threshold
1: for $keyword \in keywords$ do
2:     $l \leftarrow len(keyword)$ {number of words in $keyword$}
3:     for $s \in \{sub : sub$ is substring of $hyp \land len(sub) = l\}$ do
4:         if $dist(keyword, s) \leq t$ then
5:             return $true$
6:         end if
7:     end for
8: end for
9: return $false$
```

for the user-entered keywords. Obviously, this procedure might also have negative results. Phrases similar to the keyword, but different from it might be recognized with the same, wrong hypothesis, which would result in an increase in the false positive rate (FPR).

Note that the keywords list used by the multi-keyword classifier can be prepared during the initialization phase; therefore, this step does not influence inference phase latency. The impacts of the multi-keyword approach on memory and latency are linear with respect to the number of TTS recordings used. However, since the multi-keyword solution requires at most one additional string for each TTS recording, the memory footprint is negligible. The same reasoning can be applied to the impact on latency since Levenshtein distance calculation is much faster than ASR decoding.

## IV. Experiments Results

### A. Evaluation procedure

Our solution was tested with MOCKS 1.0 testset [32] and GSC v2 testset.

Since the AM was trained with English data, we used *en_LS_clean*, *en_LS_other*, and *en_MCV* subsets of MOCKS. Each test case is composed of two audio files and a keyword. One of those files is treated as initialization (enrollment) phase data and the other is treated as inference phase data. However, since our goal is to limit the user's interaction with the device, we skip the initialization phase data in the case of static LM. Furthermore, in the case of a multi-keyword classifier, we replace the initialization phase recording with synthetic data. We will refer to the inference phase data as test audio.

Each of the MOCKS subsets used for evaluation is split into three distinct parts:

- positive test cases – where the test audio contains a given keyword; we will call this part *pos*,
- similar test cases – where the test audio contains a different phrase than the given keyword, but both are close phonetically; we will call this part *sim*,
- different test cases – where the test audio contains a different phrase than the given keyword and the phonetic distance between both is large; we will call this part *dif*.

To present the impact of different hyperparameters on the above-described parts of MOCKS (true positive on *pos* and false positive on *sim* and *dif*), we used recognition accuracy as the main metric.

GSC is not suited for the open-vocabulary version of KWS, since it contains a very limited number of keywords. Nonetheless, we present the evaluation results of our solution on this testset for the sake of comparison with previous works. The most popular metric used with GSC is simply accuracy [33]. The negative test cases should be recognized as either `_unknown_` or `_silence_` special classes. The former contains words not included in the positive classes and the latter contains silence and non-speech events. Evaluation on GSC is a 12-class classification problem, while our solution is designed for the generic case of open-vocabulary classification. To remedy this issue evaluation for the negative test cases (labeled `_unknown_` or `_silence_`) is performed in the following way:

- In the initialization phase for each positive keyword we prepare an LM and extended keywords list (if the multi-keyword classifier is enabled).
- After audio processing is done, for each positive keyword we perform re-scoring, apply beam search, convert the result to the hypothesis, and finally compute the character level Levenshtein distance between the hypothesis and the given keyword.
- Finally, we find the minimal distance from the previous step. If this distance is less than or equal to the threshold, this sample is counted as a false positive and a true negative otherwise.

We used an internally-developed end-to-end TTS system to generate synthetic recordings for the multi-keyword classifier. The system was composed of a neural AM and a vocoder. The AM mapped sequences of phonemic labels to acoustic features, while the vocoder mapped those features to audio samples. The set of phonemic labels contained language-specific (English) symbols of phonemes, word delimiters, and end-of-sentence marks. However, during synthesis, keywords were stripped of those marks. Acoustic feature vectors were derived from F0 (interpolated in unvoiced regions), mel-spectra, and band-aperiodicity as in the case of the WORLD vocoder [34]. The vocoder architecture was based on [35] and AM was similar to the Tacotron 2 [36] architecture as described in [37], with the use of the mutual information loss (MILoss) function [38]. Audio data included in the LJ speech dataset [39] and Hi-Fi Multi-Speaker English TTS Dataset [40] were used to train the entire system. The vocoder was trained separately for each voice. The AM was trained for 10 k epochs on the entire training data, followed by 450 k epochs of each voice-specific data.

For each keyword in MOCKS and GSC, we generated 10 synthetic recordings. We chose one male and one female voice for the experiments with a multi-keyword classifier based on two synthetic recordings. Two further types of experiments with this type of classifier were performed:

1) using clean audio data;
2) using audio data mixed with background noise and convolved with RIR.

We used the same types of noise and RIR as during AM training.

For confidence interval estimation we used bootstrap resampling of the testsets. Each testset was resampled 200 times with replacement. The trainset and model remained fixed. In order to provide a 95 % confidence interval we calculated the $[2.5, 97.5]$ percentile boundaries.

## B. Evaluation results

*1) Impact of the boosting weight:* For the purpose of testing the impact of the static LM and different *boost* weights, we performed evaluations using values from the set $\{2^n : n \in \{0, 1, \ldots, 11\}\}$. Note that setting the *boost* weight to one means that none of the subword scores will be modified during re-scoring and only the AM scores will be taken into account in beam search. We treat this case as the baseline solution.

Fig. 4, 5 and 6 show acceptance rate in the function of the threshold applied in the keyword classifier for *en_LS_clean*, *en_LS_other* and *en_MCV* respectively.

Let us start the analysis of the results with $boost = 1$. For $threshold \in [0, 0.05]$, in all the testsets acceptance rates in *pos*, *sim* and *dif* are constant. This is due to the fact that all keywords in MOCKS are relatively short (phonetic transcription length $p \leq 16$). As long as the $threshold$ is greater than 0.05, acceptance rates in both *pos* and *sim* are increasing. However, those values in the former subsets grow slower than in the latter subsets. This means that increasing the threshold improves the TPR, but increases the FPR even faster. Values of acceptance rate in *dif* start to increase only with $threshold > 0.4$ since this test set contains test cases that are very different from the given keyword. For such test cases, the AM returns very different hypotheses from the keyword, even if they do not match the proper transcription. Similar

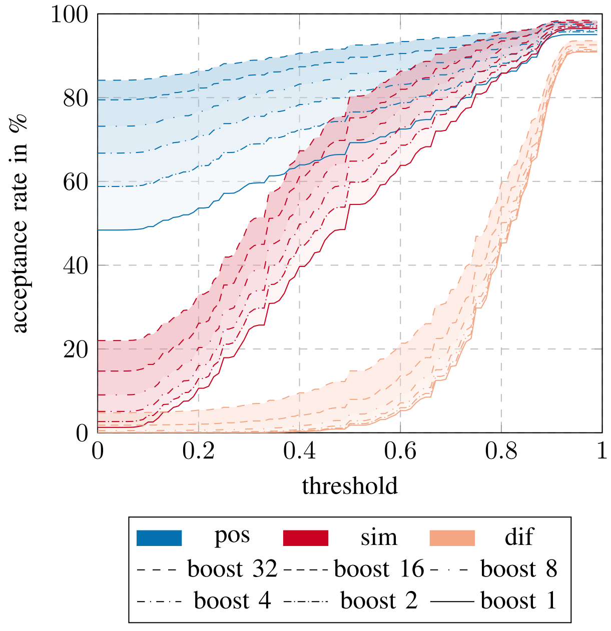

Fig. 4. Boosting results with static LM, without multi-keyword classifier, for en_LS_clean testset.

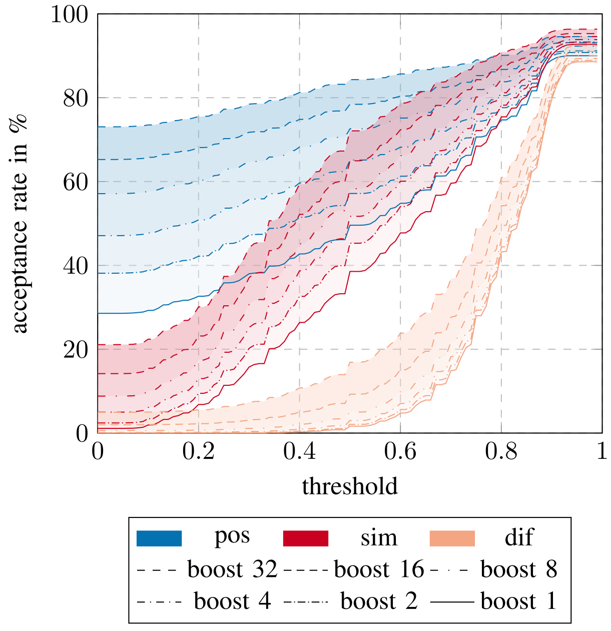

Fig. 5. Boosting results with static LM, without multi-keyword classifier, for en_LS_other testset.

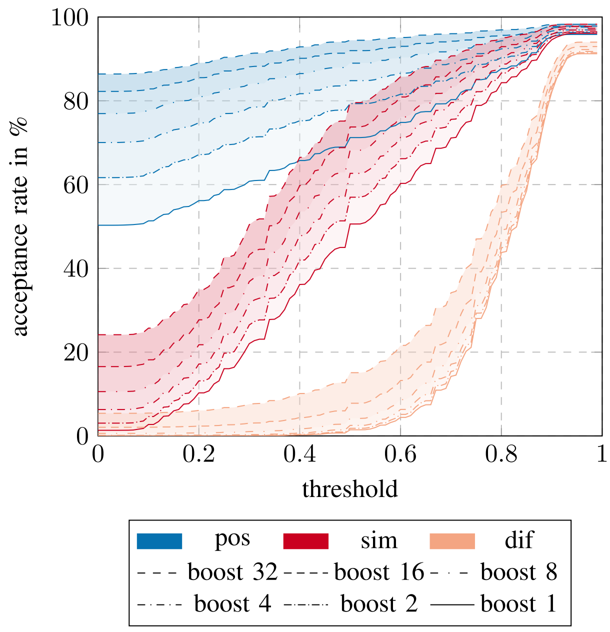

Fig. 6. Boosting results with static LM, without multi-keyword classifier, for en_MCV testset.

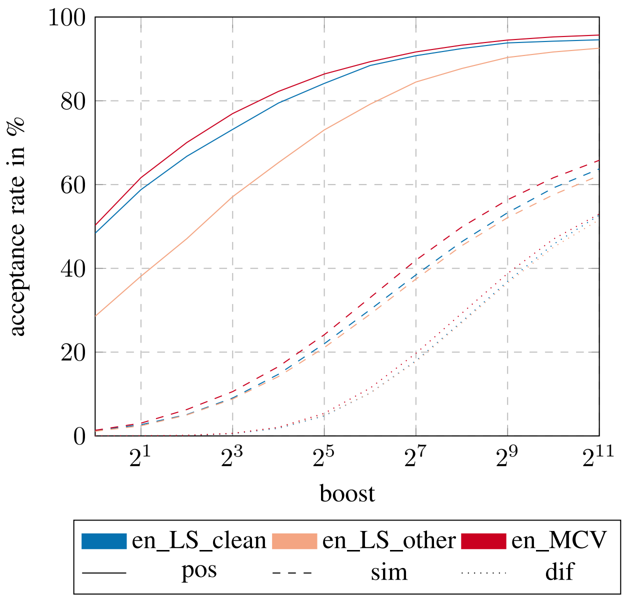

Fig. 7. Boosting results with static LM, without multi-keyword classifier, threshold 0, for MOCKS testset.

TABLE I
EER IN % FOR DIFFERENT VALUES OF BOOST ON MOCKS, WITHOUT A MULTI-KEYWORD CLASSIFIER.

| boost | *en_LS_clean* | *en_LS_other* | *en_MCV* |
|---|---|---|---|
| 1 | $30.05 \pm 0.28$ | $37.27 \pm 0.26$ | $27.30 \pm 0.16$ |
| 16 | $16.56 \pm 0.24$ | $26.04 \pm 0.23$ | $14.46 \pm 0.19$ |
| 32 | $\mathbf{15.18 \pm 0.14}$ | $22.65 \pm 0.13$ | $\mathbf{14.22 \pm 0.10}$ |
| 64 | $19.86 \pm 0.11$ | $\mathbf{20.34 \pm 0.17}$ | $21.57 \pm 0.12$ |

observations also apply to cases with *boost* $> 1$, except for *dif*, for which the higher the $threshold$, the sooner this function starts to grow. This analysis suggests that it is safe to use $threshold = 0$.

For clarity, Fig. 4, 5 and 6 show evaluation results only for $boost \leq 32$. In Fig. 7 we present evaluation results for all the testsets and $boost$ values up to 2048 using $threshold = 0$. It should be noted that for $boost \leq 32$, for all the testsets acceptance rates in *pos* are growing faster than in *sim*. These $boost$ values are also accompanied by small acceptance rates in *dif* in all the testsets. However, the larger the $boost$ gets, the faster acceptance rates in *sim* grow when compared to *pos*. This is also accompanied by a rapid growth of those rates in *dif*. This observation suggests that there is a limit for $boost$ after which static LM brings more harm than benefit.

We use an equal error rate (EER) to estimate the optimal $boost$ value. For each testset, FPR is calculated after summing *sim* and *dif* subsets. Fig. 8 shows EER for all the testsets and $boost \leq 128$ (for higher $boost$ values EER is growing hence it is omitted). The width of the lines in Fig. 8 represent confidence intervals for EER estimation. It should be noted that the minimal EER values are located at $boost$ equal to 32 or 64, depending on the testset. The rapidly growing value of EER for large $boost$ is caused by the fact that in those cases FPR is always greater than the false negative rate (FNR). Since there is no point for which FPR and FNR are equal, the largest of those values is chosen as EER. Detailed values of EER for MOCKS can be found in Table I.

Fig. 9 shows the evaluation results for different $boost$ values on the GSC testset using $threshold = 0$. Applying static LM improves accuracy from 84.60% for the baseline model to 95.97% at $boost = 32$. For higher $boost$ values accuracy drops rapidly. This is due to the fact that with those large $boost$ values the keyword subwords are favored during beam search. Therefore the number of negative test cases recognized as keywords grows. Detailed values of accuracy for GSC can be found in Table II.

*2) Impact of the multi-keyword classifier:* The main motivation for applying a multi-keyword classifier should be the increase in TPR, which ideally would not be accompanied by the increase of FPR. Our experiments show that this is not the case for MOCKS. Fig. 10 shows the evaluation results using *en_LS_clean* for multi-keyword classifiers initialized with 2 and 10 clean TTS recordings. Those results are compared to a single-keyword classifier solution. Furthermore in Table III we present exact evaluation results (acceptance rate) for each English testset in MOCKS. For clarity we limit those results to $boost = 1$ (no LM) and $boost = 32$, since this value gave the highest results as shown in Section IV-B1.

We observed that the improvement in acceptance rate on *pos* was larger than a similar increase on *sim* and *dif* only for small $boost$ values. This difference was very small for the multi-keyword classifier initialized with two synthetic recordings. However, with 10 such recordings, the improvement of the acceptance rate on *pos* was almost 2 pp. larger than on *sim*.

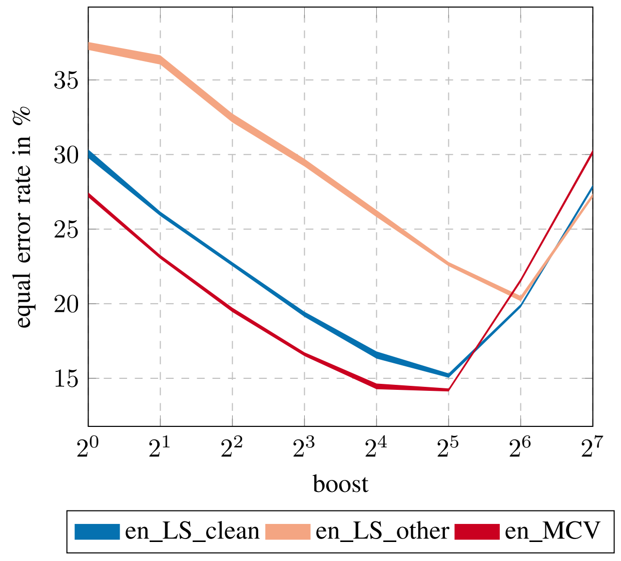

Fig. 8. Boosting results with static LM, without multi-keyword classifier, for MOCKS testset.

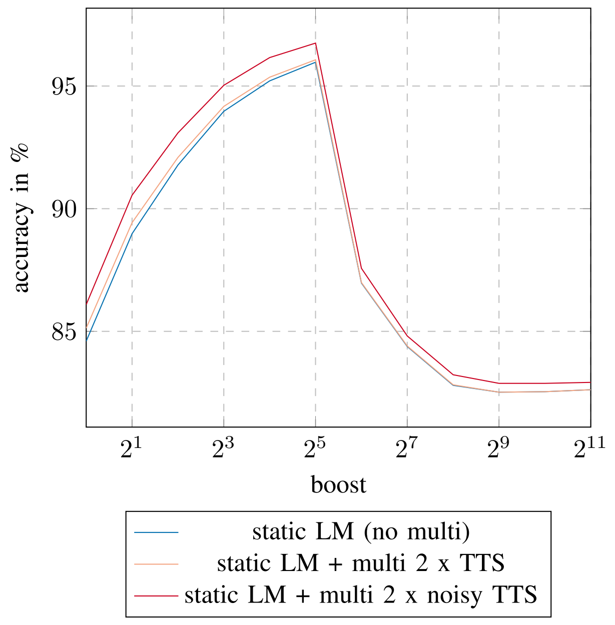

Fig. 9. Boosting results with static LM, with and without multi-keyword classifier, for GSC testset.

As soon as *boost* $= 8$ multi-keyword classifier introduced a larger increase in acceptance rate on *sim* than on *pos*, which was visible in all testsets. Mixing synthetic recordings with background noise and RIR did not improve the situation. The increase in acceptance rate on *pos* was smaller than on *sim*.

Evaluation results on GSC with a multi-classifier show a similar increase in accuracy (0.78 pp–2.3 pp depending on the *boost*). This improvement seems to be insignificant, nonetheless, it should be noted that it is gained in the range of 85 %–96 %. At this level of accuracy, even a minor increase in this metric means a substantial reduction in the number of errors.

TABLE II
ACCURACY IN % FOR DIFFERENT METHODS ON GSC.

| **method** | **accuracy** |
|---|---|
| boost 1 (baseline) | 84.60 |
| boost 1 + multi 2 x TTS | 85.13 |
| boost 1 + multi 2 x noisy TTS | 86.09 |
| boost 32 | 95.97 |
| boost 32 + multi 2 x TTS | 96.07 |
| boost 32 + multi 2 x noisy TTS | **96.75** |

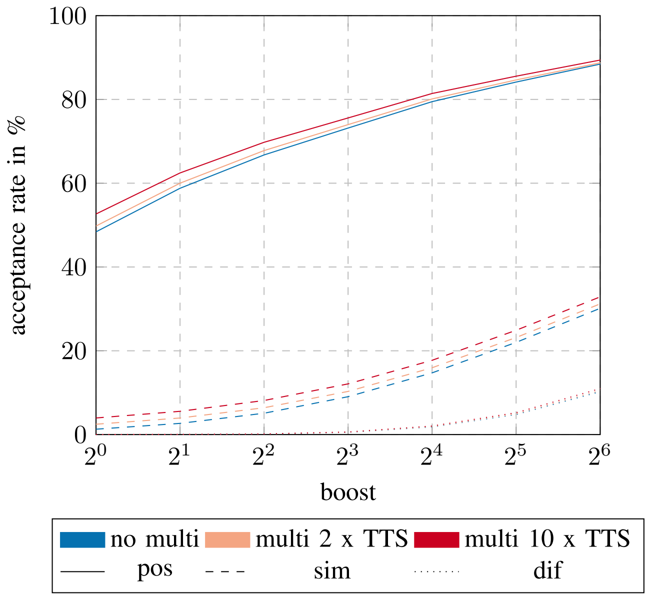

Fig. 10. Boosting results with static LM, with and without multi-keyword classifier, for en_LS_clean testset.

## V. Discussion and conclusions

Two major observations can be drawn from our experiments:

1) The increase of *boost* in static LM improves TPR, however, the larger the *boost* gets, the more dominant the increase of FPR compared to the increase of TPR.
2) The multi-keyword classifier introduces a positive impact on TPR only for very low *boost* values, while for high values of *boost*, however, the increase of FPR is much larger than the increase of TPR.

Using *boost* $= 32$ seems to be the right choice in general cases since this value resulted in the minimal EER in *en_LS_clean* and *en_MCV* and the largest accuracy in GSC. However, it should be noted that EER was minimal in *en_LS_other* with *boost* $= 64$, hence there is no universal value for this parameter.

The positive impact of the multi-keyword classifier is especially visible with the increased number of TTS recordings for each keyword. This number can be potentially unbounded, but a rule of thumb suggests using only a small amount (not exceeding 10) of such recordings. This suggestion is based on the observation that the longer the list of additional

TABLE III
ACCEPTANCE RATE IN % FOR DIFFERENT METHODS ON MOCKS.

| | **en_LS_clean** | | | **en_LS_other** | | | **en_MCV** | | |
|---|---|---|---|---|---|---|---|---|---|
| **method** | **pos** | **sim** | **dif** | **pos** | **sim** | **dif** | **pos** | **sim** | **dif** |
| boost 1 (baseline) | 48.39 | 1.28 | 0.00 | 28.56 | 1.12 | 0.00 | 50.29 | 1.34 | 0.00 |
| boost 1 + multi 2 x TTS | 49.78 | 2.47 | 0.01 | 29.98 | 1.90 | 0.02 | 51.56 | 2.38 | 0.02 |
| boost 1 + multi 10 x TTS | 52.65 | 3.96 | 0.02 | 31.89 | 3.01 | 0.03 | 52.88 | 3.70 | 0.02 |
| boost 1 + multi 2 x noisy TTS | 50.88 | 3.81 | 0.05 | 30.76 | 2.94 | 0.07 | 52.94 | 3.83 | 0.04 |
| boost 1 + multi 10 x noisy TTS | 54.80 | 7.73 | 0.11 | 34.85 | 5.89 | 0.17 | 56.22 | 7.48 | 0.11 |
| boost 32 | 84.15 | 22.01 | 4.79 | 73.05 | 21.10 | 4.96 | 86.40 | 24.16 | 5.40 |
| boost 32 + multi 2 x TTS | 84.66 | 23.17 | 4.95 | 73.46 | 21.92 | 5.12 | 86.82 | 25.44 | 5.58 |
| boost 32 + multi 10 x TTS | 85.55 | 24.90 | 5.22 | 74.45 | 23.61 | 5.46 | 87.46 | 27.10 | 5.82 |
| boost 32 + multi 2 x noisy TTS | 85.04 | 24.70 | 5.45 | 73.92 | 23.40 | 5.65 | 87.43 | 27.09 | 5.94 |
| boost 32 + multi 10 x noisy TTS | 86.54 | 29.22 | 6.24 | 76.14 | 27.46 | 6.65 | 88.52 | 31.18 | 6.59 |

keyword variants, the more likely the chance of false positive acceptance of phrases similar to the given keyword.

Evaluation of MOCKS and GSC with a multi-keyword classifier shows that there is a significant difference in both testsets. With MOCKS at $boost = 32$ adding 10 keyword variants increases FPR on *sim* more than TPR on *pos*. On the other hand with GSC at $boost = 32$ and 10 keyword variants we still observe improvement in accuracy. This can be explained by the fact that GSC contains short phrases and the negative samples are very different from the keywords in terms of character level Levenshtein distance. On the contrary, MOCKS contains longer phrases and a subset of negative samples similar to the keywords, hence they are difficult to distinguish. This means that adding additional keyword variants also increases the probability of false acceptance in this subset (*sim*). This analysis also leads to the conclusion that a multi-keyword classifier is an effective solution as long as one does not expect to deal with such challenging negative cases.

It might seem that a solution with accuracy equal to $96.75\,\%$ on GSC is far behind the current leading architecture which was evaluated on this testset and gained $98.37\,\%$ [41]. Still, it should be noted that our solution is designed for a far more complex task. GSC contains a very limited amount of keywords, all very short and distinct from each other. Finally, there are no challenging negative test cases in GSC. On the other hand, our solution is designed for the open-vocabulary case, in which the model needs to deal with keywords that are very similar to each other. Hence the evaluation results on MOCKS are much more informative and the decrease in accuracy on GSC evaluation is the cost paid for much broader generalization.

## VI. FUTURE WORK

In the future, we plan to work on methods that automatize the selection of the optimal $boost$ value. Furthermore, setting specific $boost$ values for different keywords might have positive results on evaluation results. Another intriguing research direction is using a combination of clean and noisy TTS recordings as ensembles in the multi-keyword classifier. This way it might be possible to reduce FPR without impacting TPR.

## REFERENCES

[1] I. López-Espejo, Z.-H. Tan, J. H. L. Hansen, and J. Jensen, “Deep spoken keyword spotting: An overview,” *IEEE Access*, vol. 10, pp. 4169–4199, 2022. doi: 10.1109/ACCESS.2021.3139508

[2] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[3] J. Rohlicek, W. Russell, S. Roukos, and H. Gish, “Continuous hidden markov modeling for speaker-independent word spotting,” in *International Conference on Acoustics, Speech, and Signal Processing,*, 1989. doi: 10.1109/ICASSP.1989.266505 pp. 627–630 vol.1.

[4] J. Wilpon, L. Miller, and P. Modi, “Improvements and applications for key word recognition using hidden markov modeling techniques,” in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1991)*, 1991. doi: 10.1109/ICASSP.1991.150338 pp. 309–312 vol.1.

[5] I.-F. Chen and C.-H. Lee, “A hybrid HMM/DNN approach to keyword spotting of short words,” in *Proc. Interspeech 2013*, 2013. doi: 10.21437/Interspeech.2013-397 pp. 1574–1578.

[6] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, “Multi-Task Learning and Weighted Cross-Entropy for DNN-Based Keyword Spotting,” in *Proc. Interspeech 2016*, 2016. doi: 10.21437/Interspeech.2016-1485 pp. 760–764.

[7] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural ’networks,” in *Proc. 23rd International Conference on Machine Learning (ICML 2006)*, vol. 2006, 01 2006. doi: 10.1145/1143844.1143891 pp. 369–376.

[8] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” *Advances in neural information processing systems*, vol. 28, 2015.

[9] K. Hwang, M. Lee, and W. Sung, “Online keyword spotting with a character-level recurrent neural network,” 2015.

[10] Y. Zhuang, X. Chang, Y. Qian, and K. Yu, “Unrestricted Vocabulary Keyword Spotting Using LSTM-CTC,” in *Proc. Interspeech 2016*, 2016. doi: 10.21437/Interspeech.2016-753 pp. 938–942.

[11] S. Sigtia, P. Clark, R. Haynes, H. Richards, and J. Bridle, “Multi-task learning for voice trigger detection,” in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*. IEEE, may 2020. doi: 10.1109/icassp40776.2020.9053577

[12] T. Bluche and T. Gisselbrecht, “Predicting Detection Filters for Small Footprint Open-Vocabulary Keyword Spotting,” in *Proc. Interspeech 2020*, 2020. doi: 10.21437/Interspeech.2020-1186 pp. 2552–2556.

[13] Y. He, R. Prabhavalkar, K. Rao, W. Li, A. Bakhtin, and I. McGraw, “Streaming small-footprint keyword spotting using sequence-to-sequence models,” in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2017)*, 2017. doi: 10.1109/ASRU.2017.8268974 pp. 474–481.

[14] A. Berg, M. O’Connor, and M. T. Cruz, “Keyword Transformer: A Self-Attention Model for Keyword Spotting,” in *Proc. Interspeech 2021*, 2021. doi: 10.21437/Interspeech.2021-1286 pp. 4249–4253.

[15] A. Awasthi, K. Kilgour, and H. Rom, “Teaching Keyword Spotters to Spot New Keywords with Limited Examples,” in *Proc. Interspeech 2021*, 2021. doi: 10.21437/Interspeech.2021-1395 pp. 4254–4258.

[16] M. Mazumder, C. Banbury, J. Meyer, P. Warden, and V. J. Reddi, "Few-Shot Keyword Spotting in Any Language," in *Proc. Interspeech 2021*, 2021. doi: 10.21437/Interspeech.2021-1966 pp. 4214–4218.

[17] L. Lugosch, S. Myer, and V. S. Tomar, "Donut: Ctc-based query-by-example keyword spotting," *arXiv preprint arXiv:1811.10736*, 2018.

[18] B. Kim, M. Lee, J. Lee, Y. Kim, and K. Hwang, "Query-by-example on-device keyword spotting," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU 2019)*, 12 2019. doi: 10.1109/ASRU46091.2019.9004014 pp. 532–538.

[19] J. Huang, W. Gharbieh, H. S. Shim, and E. Kim, "Query-by-example keyword spotting system using multi-head attention and soft-triple loss," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2021)*, 2021. doi: 10.1109/ICASSP39728.2021.9414156 pp. 6858–6862.

[20] J. Huang, W. Gharbieh, Q. Wan, H. S. Shim, and H. C. Lee, "QbyE-MLPMixer: Query-by-Example Open-Vocabulary Keyword Spotting using MLPMixer," in *Proc. Interspeech 2022*, 2022. doi: 10.21437/Interspeech.2022-11080 pp. 5200–5204.

[21] S. Settle, K. Levin, H. Kamper, and K. Livescu, "Query-by-Example Search with Discriminative Neural Acoustic Word Embeddings," in *Proc. Interspeech 2017*, 2017. doi: 10.21437/Interspeech.2017-1592 pp. 2874–2878.

[22] G. Chen, C. Parada, and T. N. Sainath, "Query-by-example keyword spotting using long short-term memory networks," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015)*, 2015. doi: 10.1109/ICASSP.2015.7178970 pp. 5236–5240.

[23] C. Chiu and C. Raffel, "Monotonic chunkwise attention," *CoRR*, vol. abs/1712.05382, 2017. [Online]. Available: http://arxiv.org/abs/1712.05382

[24] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, p. 1789–1819, jun 2021. doi: 10.1007/s11263-021-01453-z

[25] S. Kim, T. Hori, and S. Watanabe, "Joint ctc-attention based end-to-end speech recognition using multi-task learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*, 03 2017. doi: 10.1109/ICASSP.2017.7953075 pp. 4835–4839.

[26] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015. [Online]. Available: http://arxiv.org/abs/1508.07909

[27] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*, 2017. doi: 10.1109/ICASSP.2017.7952261 pp. 776–780.

[28] D. Snyder, G. Chen, and D. Povey, "Musan: A music, speech, and noise corpus," *arXiv preprint arXiv:1510.08484*, 2015.

[29] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," in *Proc. Interspeech 2019*, 2019. doi: 10.21437/Interspeech.2019-2680 pp. 2613–2617.

[30] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015)*, 2015. doi: 10.1109/ICASSP.2015.7178964 pp. 5206–5210.

[31] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common voice: A massively-multilingual speech corpus," in *International Conference on Language Resources and Evaluation*, 2019.

[32] M. Pudo, M. Wosik, A. Cieślak, J. Krzywdziak, B. Łukasiak, and A. Janicki, "MOCKS 1.0: Multilingual open custom keyword spotting testset," in *Proc. Interspeech 2023*, in press.

[33] "Keyword spotting on google speech commands," https://paperswithcode.com/sota/keyword-spotting-on-google-speech-commands, 2023, [Online; accessed 19-May-2023].

[34] M. Morise, F. Yokomori, and K. Ozawa, "World: A vocoder-based high-quality speech synthesis system for real-time applications," *IEICE Transactions on Information and Systems*, vol. E99.D, pp. 1877–1884, 07 2016. doi: 10.1587/transinf.2015EDP7457

[35] J.-M. Valin and J. Skoglund, "A Real-Time Wideband Neural Vocoder at 1.6kb/s Using LPCNet," in *Proc. Interspeech 2019*, 2019. doi: 10.21437/Interspeech.2019-1255 pp. 3406–3410.

[36] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, R. A. Saurous, Y. Agiomvrgiannakis, and Y. Wu, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*, 2018. doi: 10.1109/ICASSP.2018.8461368 pp. 4779–4783.

[37] N. Ellinas, G. Vamvoukakis, K. Markopoulos, A. Chalamandaris, G. Maniati, P. Kakoulidis, S. Raptis, J. S. Sung, H. Park, and P. Tsiakoulis, "High Quality Streaming Speech Synthesis with Low, Sentence-Length-Independent Latency," in *Proc. Interspeech 2020*, 2020. doi: 10.21437/Interspeech.2020-2464 pp. 2022–2026.

[38] P. Liu, X. Wu, S. Kang, G. Li, D. Su, and D. Yu, "Maximizing mutual information for tacotron," *ArXiv*, vol. abs/1909.01145, 2019.

[39] K. Ito and L. Johnson, "The LJ speech dataset," https://keithito.com/LJ-Speech-Dataset/, 2017.

[40] E. Bakhturina, V. Lavrukhin, B. Ginsburg, and Y. Zhang, "Hi-Fi Multi-Speaker English TTS Dataset," in *Proc. Interspeech 2021*, 2021. doi: 10.21437/Interspeech.2021-1599 pp. 2776–2780.

[41] R. Vygon and N. Mikhaylovskiy, "Learning efficient representations for keyword spotting with triplet loss," in *Speech and Computer*, A. Karpov and R. Potapova, Eds. Cham: Springer International Publishing, 2021. ISBN 978-3-030-87802-3 pp. 773–785.