IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 23 May 2024, accepted 20 June 2024, date of publication 1 July 2024, date of current version 9 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3421605

APPLIED RESEARCH

Improved Small-Footprint ASR-Based Solution for Open Vocabulary Keyword Spotting

MIKOŁAJ PUDO[1,2], MATEUSZ WOSIK[1], AND ARTUR JANICKI[2], (Member, IEEE)
[1]Samsung Research and Development Institute Poland, 31-416 Kraków, Poland
[2]Faculty of Electronics and Information Technology, Warsaw University of Technology, 00-661 Warsaw, Poland

Corresponding author: Mikołaj Pudo (m.pudo@samsung.com)

ABSTRACT This article presents an improved solution to the open vocabulary keyword spotting task with the keyword given by text. This solution is based on the acoustic model and unigram language model architectures. Such models are commonly used in the automatic speech recognition task. However, they can also be minimized and deployed to mobile devices while preserving the ability to transcribe extensive vocabulary data. Our improvements can be applied to any type of sequence-to-sequence model architecture generating token probabilities. Furthermore, they do not increase the latency since they are applied to the acoustic model output. We propose three modifications: 1) leveraging multiple hypotheses generated by the beam search algorithm, 2) modifying the method of the language model initialization, and 3) smoothing the acoustic model outputs. We evaluated those improvements on the public testsets (MOCKS: Multilingual Open Custom Keyword Spotting Testset and Google Speech Commands) with an exemplary highly compressed acoustic model. Comparison of the results with the baseline solution revealed an equal error rate reduction by 1.6–1.7% relative depending on the testset.

INDEX TERMS Acoustic model, custom keyword spotting, language model, on-device keyword spotting, open vocabulary keyword spotting, query-by-text.

I. INTRODUCTION

In recent years, keyword spotting (KWS) has gained significant attention. While traditional KWS involves a limited set of predefined keywords, the open vocabulary version of this task poses a more complex challenge of recognizing any word or phrase. This task is essential for various applications such as voice assistants, transcription systems, audio search engines, and more, where detecting and understanding specific keywords in real time is crucial. Throughout this paper, KWS will refer to the open vocabulary version of this task.

Solving the KWS problem requires overcoming several constraints and challenges. One such constraint is the necessity of running solutions on-device. KWS is usually performed in real time as the initial step of the processing pipeline. Furthermore, with an increasing emphasis on protecting user data, processing audio locally on the user's device becomes imperative. Thus, on-device execution is meant to secure low latency and avoid transmitting sensitive information to the cloud. This constraint introduces new challenges as deploying complex keyword recognition models without compromising the system's functionality on resource-constrained mobile devices like smartphones and IoT devices becomes daunting.

Another significant challenge in KWS is distinguishing similar words from the target keywords. Words with similar phonetic properties or homophones pose considerable difficulty in accurate recognition. Resolving this issue requires advanced solutions that can contextually understand the audio input and correctly interpret the intended word or phrase.

Overall, addressing the constraints and challenges in KWS requires striking the right balance between the accuracy and efficiency of the system. This article proposes a solution based on the acoustic model (AM) and language model (LM). AMs commonly used in the large vocabulary automatic speech recognition (ASR) task are too large to run on-device [1]. There are techniques for compressing AMs, such

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Fiumara.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

as knowledge distillation (KD) [2], [3], [4], [5], where the goal is to train a model with a reduced number of parameters by extracting information stored in a full-scale model with a similar architecture. Another technique applied to reduce the model size is quantization [6], [7], [8], [9], which is based on the idea of replacing floating point model parameters with lower precision numbers. However, model compression usually results in a drop in the quality of hypotheses [10]. A simple unigram LM can fix this issue without impacting the latency [11]. In this article, we build upon this idea. Our modifications to the baseline method improve the results on multiple testsets without increasing the latency. Our contribution described in this article is the following:

- We propose extending the hypotheses set compared with the keyword with the top-k beam path outputs during inference.
- We propose an improved method of LM weights initialization.
- We introduce AM output scores smoothing.

Improvements presented in this article can be easily applied to KWS solutions based on the models processing audio data and generating per-frame probability distributions. In the most straightforward case, the sample space consists of tokens in the given vocabulary (e.g., phonemes, letters, or strings). The model output can be decoded either with a simple greedy search or variants of the beam-search algorithm [12]. In both cases, re-scoring is easy to apply, and the simplicity of the unigram LM ensures a minor impact on the computational complexity.

The rest of this paper is organized as follows. Section II discusses previously implemented solutions to the KWS problem. In Section III, we present baseline model architecture with its respective parts (Subsection III-A) and the proposed improvements (Subsection III-B). Results of experiments with those improvements are presented in Section IV. We conclude this paper with a discussion in Section V and provide some possible future research directions in Section VI.

II. RELATED WORK

KWS can be categorized into query-by-text (QbyT) and query-by-example (QbyE). In QbyT, the keyword is obtained from written text, while in QbyE, one or more audio recordings are provided during the initialization phase to serve as ''enrollment'' examples. In the inference stage, these recordings are utilized as ''keywords'' in various ways depending on the chosen solution.

Most QbyE solutions are implemented on an acoustic level by comparing embeddings generated by the enrollment and test audio data. Such an idea was the basis of the solutions presented in [13], [14], [15], [16], [17], and [18], where sequence-to-sequence models were used to generate embeddings for the enrollment and test audio data. Different metrics (e.g., cosine similarity) were applied to those embeddings, and the distance was compared with a predefined threshold. An interesting variant of this approach was used in [19] and [20], where an additional classification layer was used on top of the embedding model. This layer was adapted for a specific keyword with the enrollment data. QbyE can also be approached as a few-shot learning keyword spotting task (FS-KWS) when the target keyword is not seen during training. Applying this perspective [21] introduced an unsupervised training approach that used only synthetic data. The goal was to alleviate the requirement of a large labeled dataset containing target keywords to learn rich embedding space representation.

A large group of QbyE solutions can be characterized as converting QbyE to QbyT. Such solutions operate on the textual output of the model rather than directly on the acoustic-level embeddings. ASR model with connectionist temporal classification (CTC) [22] was used in [23] to generate n-best phonetic level keyword phrases for the enrollment audio. During inference, each test audio was processed with a similar AM. The log probability was computed for each keyword, weighted by respective confidence score, and added to the final score. The keyword was detected if this score was above a certain predefined threshold. A similar approach was also used in [24]. An ASR model processed enrollment audio data, and phonetic level posteriorgrams were compiled into finite-state transducers (FST). In the inference phase, the audio was processed with the AM, and the output was scored using the keyword model FST. Finally, the score was compared with the threshold, which was chosen automatically based on the enrollment recordings and negative samples generated by rearranging each enrollment waveform.

Unlike QbyE, which needs multiple audio samples, QbyT uses text to provide the keyword. A thorough review of solutions to this problem can be found in [25]. The models designed for this task have evolved similarly to the acoustic models used in speech recognition. Previously, the models were based on the hidden Markov model (HMM) and Gaussian mixture model (GMM) architecture [26], [27]. The GMM modeled the acoustic features in this approach, and its output was used as the HMM emission probabilities. The Viterbi decoding [28] was used for inference to find the best path in the decoding graph generated by HMM. Later, the GMM component was replaced by deep neural networks (DNN) [29], [30]. With the advent of end-to-end architectures in speech processing, techniques such as CTC and attention mechanisms [31] have also become standard solutions in KWS. These models typically consist of an encoder and a decoder. The encoder generates frame-level embeddings, while the decoder is trained to compile those embeddings into a sequence of posterior probabilities, which are usually much sparser than the frame sequence. The solutions differ in how the decoder generates the posteriors and how they are handled in the postprocessing step. Additionally, some solutions have a more sophisticated transformation in the keyword encoder module, while others use a simple identity function or translation to phonetic transcription.

The most straightforward methods in QbyT just use the output from the ASR model with CTC. One such model, described in [32], consisted of three long short-term memory (LSTM) layers that predicted characters. When the negative log posterior of the output keyword character sequence was below a predefined threshold then the keyword was detected. A similar architecture based on the LSTM-CTC model was used in [33] but with phonetic vocabulary. The model hypothesis was compared with minimum edit distance to one or more phoneme sequences representing the keyword during inference. Detection occurred when the distance was lower than a predefined threshold, estimated separately for each phoneme keyword sequence based on the training data and lexicon.

The connection between KWS and ASR can also be limited to the training phase. In [34], the model was trained using a multi-task approach incorporating ASR and KWS outputs. During inference, the model relied solely on the KWS output. This approach aimed to enhance the model's generalizability and performance in challenging acoustic environments. However, multi-task training with a KWS target focused the model on a single keyword; thus, it lacked open vocabulary support. The solution outlined in [35] utilized an audio encoder network and a convolutional classifier. The audio encoder network was trained using the ASR task, while the classifier network employed filters generated by a keyword encoder. The keyword encoder was a bidirectional LSTM (BiLSTM) layer that processed the text keyword provided by the user. In [36], an ASR model composed of five LSTM layers was utilized. While the model was trained with CTC loss, it also incorporated a keyword encoder and attention network to direct the prediction network's focus toward the specific keyword of interest. One of the models described in the paper adopted a phoneme level 6-gram LM ($\sim$ 1.5 M 6-grams), which enhanced the model's performance.

There were also multiple propositions based on joint audio and text embeddings. In [37], input audio queries were compared with enrolled text keyword sequences. An adapted attention-based cross-model matching approach placed the audio and text representations within a shared latent space. It was trained end-to-end with monotonic matching loss and keyword classification loss. Another audio-text-based end-to-end model solution for KWS was presented in [38]. The model consisted of an audio encoder with conformer architecture [39] and a text encoder to get respective embeddings. A projection block was used to map those individual embeddings to a common latent space where they were aligned using the dynamic sequence partitioning (DSP) algorithm to make a prediction. The DSP algorithm was also used with cosine similarity in the classifier network in [40]. A small-footprint conformer model trained with CTC loss on a rich speech dataset was employed as an audio encoder. The text encoder converted the target keyword to phonemes applying a grapheme-to-phoneme (G2P) model and then to phoneme vectors using a phoneme-to-vector database built beforehand with an audio encoder. Audio and phoneme embeddings were used in the classifier network to make the prediction.

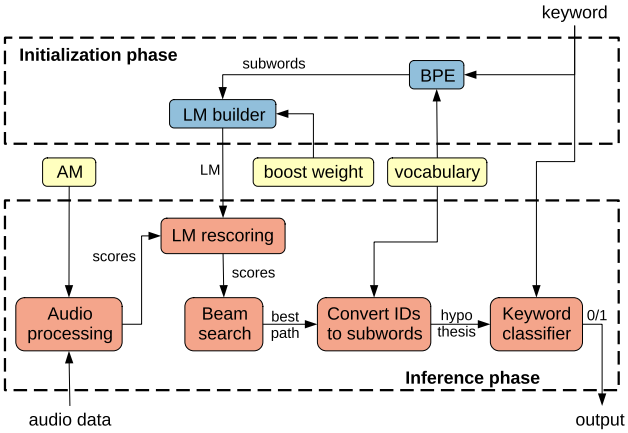

FIGURE 1. Overview of the baseline KWS solution with static LM [11].

An interesting solution named AdaKWS was presented in [41]. It was also based on an audio and text encoder with a classification layer. The frozen Whisper [42] encoder model was used to process audio data, allowing robust generalization in various acoustic conditions. The text encoder was trained to output keyword-conditioned normalization parameters. The keyword-conditioned audio representation was fed to the linear classifier layer to decide whether the keyword was present. This solution achieved state-of-the-art results on KWS benchmarks but required higher computational resources so that the on-device usage might be limited.

It should be noted that many KWS solutions were evaluated with a subset of Google Speech Commands (GSC) [43]. It is a public dataset developed for training and evaluating models designed for simple command recognition. It is also used for the KWS task. Despite its popularity, the GSC testset cannot evaluate KWS solutions thoroughly for several reasons: the number of keywords is relatively low, and the negative samples are entirely different from the keywords. All the samples were recorded without background noise and were cut with high precision. These issues make GSC inadequate for emulating actual production conditions. However, GSC is a useful testset for classification problems with a fixed number of classes.

III. MODEL ARCHITECTURE

The solution presented in this paper is an extension of the KWS architecture proposed in [11]. The overview of the baseline solution is shown in Figure 1. It comprises three main elements: AM, LM, and a keyword classifier. The AM is used to generate frame-level scores for each output token. These scores are subsequently weighted by the values stored in the LM. Finally, the beam search algorithm is used to obtain the best path, which is converted into text transcription of the phrase in the audio data. Such a solution is commonly used in ASR systems. However, the KWS system should be designed as a binary classifier; hence, the output from

TABLE 1. AM architecture.

component	teacher	student
input	power-mels, 40-dim, 25 ms window, 10 ms step, cepstral mean and variance normalization	
	encoder	
BiLSTM	2 x 512	2 x 124
max-pooling	✓	✓
BiLSTM	2 x 512	2 x 124
max-pooling	✓	✓
BiLSTM	2 x 512	2 x 124
max-pooling	✓	✓
BiLSTM	2 x 512	2 x 124
BiLSTM	2 x 512	2 x 124
BiLSTM	2 x 512	2 x 124
	decoder	
LSTM	1 000	256
embedding	621	156
readout	1 000	256
soft attention	512	124
hard attention	512	124
chunk size	2	2
output	500 tokens	
parameters	50.1 M	3.1 M

the ASR module is compared with the given keyword, and based on their similarity, the True or False value is returned. We describe each of those modules in detail below.

A. BASELINE ARCHITECTURE

1) ACOUSTIC MODEL (AM)

The AM used in the experiments described below was based on a sequence-to-sequence architecture implementing a monotonic chunkwise attention mechanism (MoChA) [44]. This approach splits the audio data into frames for which the model input features are computed. The encoder generates frame-level embeddings, which are processed by the decoder equipped with the attention mechanism. The output from the decoder consists of a sequence of vectors interpreted as the likelihood of the output tokens. Such tokens are usually extracted from the training data and consist of phonemes, letters, word parts, or entire words.

Since KWS solutions are usually designed to run on customer devices, the goal is to create a small model while preserving high performance. Hence, we applied KD to compress the model, as in [45]. There are two steps in this paradigm:

1) training a large teacher model,
2) training a small student model leveraging the knowledge obtained by the teacher.

In our solution, both models shared the same architecture but differed in layer sizes. A detailed description of teacher and student models is presented in Table 1. The input normalization parameters were computed over all training samples. The model's output vocabulary was generated using the adaptation of byte pair encoding (BPE) [46] over all transcriptions from the trainset.

TABLE 2. Hyperparameters of AM training procedure.

component	teacher	student
loss	CTC + cross-entropy	distillation (0.4) + cross-entropy (0.6)
training length	23 epochs	22 epochs
valid. period	5 000 steps	
initial LR	1×10^{-4}	4×10^{-4}
LR decay	0.95	
LR schedule	apply decay every 10 validation steps without change	
trainset	LibriSpeech + MCV + 4 h non-speech data	
trainset augmentation	noise + RIR + SpecAugment [50]	noise + RIR
quantization	none	8-bit post-training

Table 2 describes the training procedure for teacher and student models. We used all the training splits from LibriSpeech [47] and all the samples from Mozilla Common Voice (MCV) version 7.0 [48], which were not included in the testset. The data was mixed with background noise from AudioSet [49] (with random signal-to-noise ratio in the range −2 dB–12 dB) and augmented with randomly selected in-house room impulse response (RIR). The RIR dataset contained simulations of distances from one to five meters and reverberation time between 0.2 s–0.9 s. We applied exponential learning rate (LR) decay every 10 validation steps without change.

The final model size was approximately 3.2 MB, and the model scored the following word error rates (WER): 16.0 % on LibriSpeech *test-clean* and 30.9 % on LibriSpeech *test-other*.

2) LANGUAGE MODEL (LM)

We used a simple unigram LM to re-score the token likelihoods generated by the AM, where the model is a vector of the same length as the AM output layer size. This type of LM introduces only a minor additional memory footprint and a slight 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.

The initialization of weights 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 used a straightforward initialization method which we named Static LM. In this method, LM weights were initialized only with two values: 1 and *boost* weight, which was treated as a model hyperparameter. The BPE algorithm was applied to the keyword with the same vocabulary that was used to convert the AM scores to obtain the hypothesis. Tokens included in the keyword were assigned a *boost* weight, and all the remaining tokens were assigned 1. Note that setting *boost* weight to 1 would not change AM scores, and setting *boost* weight to values smaller

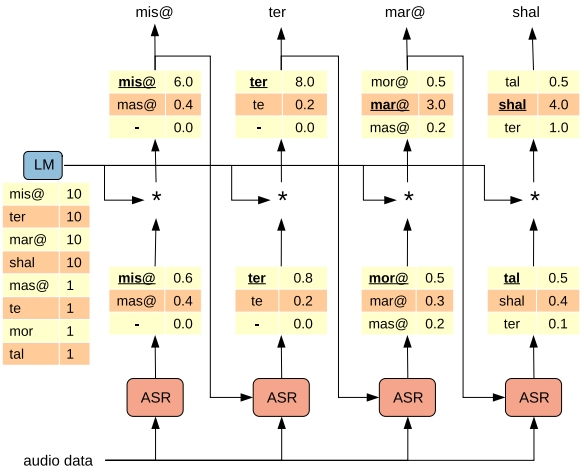

FIGURE 2. LM re-scoring example. ASR consists of AM and beam search (width = 3). A star denotes multiplication by LM weights.

than one would decrease the probability of recognizing the keyword.

Figure 2 shows an example application of the LM re-scoring. The processing flow is from bottom to top for each token and from left to right for consecutive tokens. In this example, ASR consists of the AM and the beam search algorithm, with a beam width equal to 3. Each ASR output is re-scored by the LM. The correct keyword is "`mister marshall`". LM is initialized so that the tokens included in the keyword are assigned weight 10, and all the remaining tokens are assigned weight 1. Without LM re-scoring, the hypothesis would be "`mister mortal`". However, re-scoring modifies the token scores and forces the model to return the keyword as the hypothesis.

3) KEYWORD CLASSIFIER

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

Algorithm 1 Keyword Classifier Algorithm

Input: *keyword* – custom keyword
Input: *hyp* – hypothesis returned by AM
Input: t – recognition threshold
$L \leftarrow len(keyword)$ {number of words in *keyword*}
for $s \in \{sub : sub \text{ is substring of } hyp \wedge len(sub) = L\}$ **do**
  if $dist\_norm(keyword, s) \leq t$ **then**
    return *true*
  end if
end for
return *false*

B. PROPOSED IMPROVEMENTS

1) ALL BEAM PATHS – MULTI-HYPOTHESES

The concept of the solution proposed in this paper is based on generating a hypothesis from an ASR model and comparing it with the given keyword. Contemporary ASR models employ the beam search algorithm to create the hypothesis. The beam width is a parameter that directly impacts the latency, memory footprint, and quality of the hypothesis. A beam width of 4 was used in the baseline solution, but only the best path was compared with the keyword. However, multiple hypotheses can be compared with the given keyword, and the one with the smallest distance can be used for further processing. This idea requires extending the keyword classifier in such a way that it would iterate over the list of hypotheses and search for the one with minimal edit distance from the keyword. The final decision should be made by comparing this distance with a threshold. The list of hypotheses can be simply composed of all the paths returned by the beam search algorithm.

For example, let the keyword be "`mister marshall`", and assume that the audio sample contains this keyword (positive test case). Let the hypotheses generated by the AM and LM be (in order of the score):

1) "`mr martial`",
2) "`mister marshall`",
3) "`mister martial`".

Normalized character level Levenshtein distances to the keyword for those hypotheses would be 0.467, 0.000, and 0.200, respectively. The second beam has the lowest distance, which is also the correct hypothesis. This test case would fail if only the best beam would be considered. However, analyzing all beams fixes this test case.

This solution will be referred to as "all beams", while the baseline solution will be called "best beam".

2) LANGUAGE MODEL WITH NEIGHBOR TOKENS WEIGHT BOOSTING

It has been shown that re-scoring token weights with a unigram LM can significantly improve the results in terms of EER [11]. However, with large boost values, the tokens contained in the keyword dominate over other tokens. This can be problematic with phrases similar to the keyword but slightly different. In this case, even if the AM assigns high values to the correct tokens, re-scoring can set the preference for the incorrect tokens.

For example:

- let the keyword be "`an action`", with token split `(an, ac-, tion)`,

- let the transcription of the audio sample be "+in action+" (negative test case), with the sequence of tokens correctly recognized by the AM +(in, ac-, tion)+.

With the LM boosting tokens contained in the keyword, it might happen that token an will be preferred by beam search, causing false acceptance.

To mitigate this issue, we propose to extend the list of boosted tokens. The idea is to assign large boost values to the tokens contained in the keyword and moderate boost values to tokens that could be included in phrases similar to the keyword.

Listing all possible phrases similar to the arbitrary keyword might be challenging. A useful heuristic that can be used to remedy this issue is the following:

$$w(s) = \begin{cases} boost & \text{if } s \in BPE(keyword), \\ n\_boost & \text{if } s \notin BPE(keyword) \wedge \\ & \exists_{t \in BPE(keyword)} dist(s,t) = 1, \\ 1 & \text{otherwise.} \end{cases} \quad (1)$$

In this equation, $w(s)$ is the weight assigned to token s during re-scoring, $BPE(keyword)$ is the set of tokens included in the keyword (generated by the BPE algorithm), $dist(s,t)$ is the character-level Levenshtein distance. Finally, *boost* and *n_boost* scalar values are the hyperparameters of the LM. The latter is used to re-score tokens similar to those contained in the keyword. Setting $n\_boost > 1$ increases the scores of those tokens; hence, this method will be referred to as "neighbor boosting".

3) OUTPUT SMOOTHING

The KWS solutions are usually deployed on devices with limited resources; thus, the model must be highly compressed. As a result, the model, which should support any type of keyword, has limited memorization capabilities. This limitation hampers the model's ability to generalize. In particular, this is the case with AMs and tokens that were infrequent during the training. Consequently, the model returns similar, more "popular" tokens for rare phrases with high confidence during inference. Despite employing various regularization techniques during the training, the overconfidence persists.

To address the issue of generating incorrect tokens, we found that higher boosting weight in the LM was necessary to counteract the AM's overconfidence [11]. However, as mentioned in Section III-B2, increasing this weight too high had an adverse impact on negative test cases. The model became overconfident with tokens contained in the given keyword. To mitigate this problem, we propose inference-time smoothing of the model's outputs. We will refer to this method as "output smoothing".

This proposition is based on label smoothing [51], a widely used regularization technique that helps prevent neural networks from becoming overconfident and improves their generalization capabilities. Typically, it is applied during training by replacing ground-truth, one-hot distribution y_{hot} with:

$$y_{ls} = (1-\alpha) * y_{hot} + \alpha/K \quad (2)$$

Here K represents the number of label classes, and α is a hyperparameter determining the amount of smoothing. Such smoothed ground-truth distribution is used to compute the loss.

In output smoothing, we modify the model output probabilities y_{pr} by:

$$y_{ls}(k) = \begin{cases} y_{pr}(k) * (1-\alpha) & \text{if } k = argmax(y_{pr}), \\ y_{pr}(k) + \alpha \dfrac{max(y_{pr})}{|vocab| - 1} & \text{otherwise.} \end{cases} \quad (3)$$

Here $|vocab|$ is the model vocabulary size, and $\alpha \in [0, 1]$ is once more a hyperparameter determining the amount of smoothing. Note that this method reduces the highest score by factor α and distributes this amount equally among all the other tokens.

IV. EXPERIMENT RESULTS

A. EVALUATION PROCEDURE

Our solution was tested with Multilingual Open Custom Keyword Spotting Testset 1.0 (MOCKS) [52] and GSC v2 testset, and followed the same evaluation procedure as in [11]. Each test case consisted of a keyword given by text and an audio file. We used only English subsets of MOCKS (*en_LS_clean*, *en_LS_other*, and *en_MCV*). Each of those subsets was split into three distinct parts:

- positive test cases – where the test audio contained a given keyword,
- similar test cases – where the test audio contained a different phrase than the given keyword, but both were close phonetically,
- different test cases – where the test audio contained a different phrase than the given keyword, and the phonetic distance between both was large.

We used the equal error rate (EER) metric to compare the impact of the proposed features with respect to their hyperparameters. EER was computed for each MOCKS split separately.

Although GSC was not explicitly designed for the open vocabulary KWS task, we employed it to contrast our proposed improvements with previous studies. We presented our findings using accuracy, a widely employed metric in this testset. Evaluation on GSC is a 12-class classification problem (10 positive classes and two negative classes: _silence_ and _unknown_). At the same time, our solution was designed for the generic case of open vocabulary classification. To compute accuracy on this testset, we proceeded as follows:

- Each positive test case was counted as 1 if the reference was contained in any of the hypotheses generated by AM and LM, and it was counted as 0 otherwise.

TABLE 3. EER in % for different combinations of proposed improvements on en_LS_clean.

beams	n_boost	α	boost	EER [%]
best	1	0.0	36	14.87 ± 0.06
all	1	0.0	13	13.64 ± 0.08
all	1	0.1	12	13.82 ± 0.07
all	4	0.0	25	13.17 ± 0.09
all	3	0.1	20	13.30 ± 0.11

TABLE 4. EER in % for different combinations of proposed improvements on en_LS_other.

beams	n_boost	α	boost	EER [%]
best	1	0.0	68	20.17 ± 0.11
all	1	0.0	25	18.51 ± 0.03
all	1	0.1	22	18.55 ± 0.20
all	6	0.0	58	17.85 ± 0.10
all	4	0.1	42	18.12 ± 0.02

TABLE 5. EER in % for different combinations of proposed improvements on en_MCV.

beams	n_boost	α	boost	EER [%]
best	1	0.0	28	13.96 ± 0.08
all	1	0.0	10	12.69 ± 0.14
all	1	0.1	9	12.93 ± 0.08
all	4	0.0	22	12.36 ± 0.06
all	4	0.1	20	12.63 ± 0.06

- Each test case from `_unknown_` and `_silence_` special classes was counted as 0 if any of the hypotheses generated by AM and LM contained any of the positive keywords and it was counted as 1 otherwise.

Accuracy was computed as the average over the abovementioned values.

For confidence interval estimation, we used bootstrap resampling of the testsets [53]. The trainset and model remained fixed during the evaluation. Each testset was resampled 200 times with replacement, and an evaluation was performed with such data. To provide a 95 % confidence interval, we calculated the [2.5, 97.5] percentile boundaries over all resampled evaluation results.

B. EVALUATION RESULTS

Detailed results for each method and subset of MOCKS can be found in Tables 3, 4, and 5. These tables contain EER results for the best hyperparameter values found in our experiments. The baseline solution used only the best beam, no neighbor token boosting ($n\_boost = 1$), and no output smoothing ($\alpha = 0.0$). Hence, there was only one hyperparameter: *boost*. We searched for the minimal EER over the set of integer boost values since such a level of granularity is a fair compromise between precision and compute resources necessary to perform the experiments. The baseline experiment results are presented in the top rows of each table.

Similarly, Table 6 contains detailed results of evaluations on GSC in terms of accuracy. Once again, the baseline model is presented in the top row.

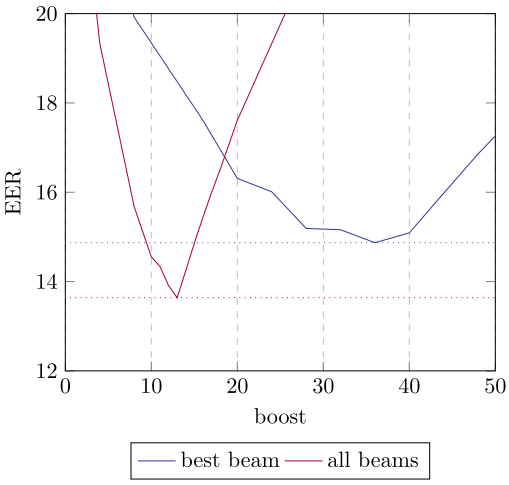

FIGURE 3. Evaluation results with unigram LM, using hypothesis from the best beam compared with hypotheses from all beams, for en_LS_clean testset.

TABLE 6. Accuracy in % for different combinations of proposed improvements on GSC.

beams	n_boost	α	boost	Accuracy
best	1	0.0	31	95.97 ± 0.38
best	2	0.0	54	96.07 ± 0.42
best	1	0.1	28	95.93 ± 0.45
all	1	0.0	1	85.64 ± 0.22
all	6	0.0	6	93.46 ± 0.62
all	1	0.1	1	89.63 ± 0.25

1) ALL BEAM PATHS IMPACT

The first modification we evaluated was extending the list of hypotheses compared with the keyword with other beams. Once again, we searched for the minimal EER over integer boost values. Figures 3 and 4 present the comparison of the baseline model (best beam) with the proposed modification (all beams) for en_LS_clean in MOCKS and GSC, respectively.

We observed that in the case of MOCKS, this simple trick lowered the minimal EER by more than 1% relative for each subset. Furthermore, with all beams being compared with the keyword, it was possible to significantly reduce the boost value for which the minimal EER was obtained. We also observed that EER in the function of boost had a more significant gradient in the case of "all beams" than with the "best beam". Hence, boost values with low EER were more specific for different MOCKS subsets with "all beams". On the other hand, it was easier to choose one boost value, which gave low EER results for all the subsets with "best beam".

Comparison of the "best beam" and "all beams" methods on GSC proved that the accuracy dropped in the latter case (Figure 4). Detailed analysis of the results showed that the accuracy increased in the positive test cases with large boost values. However, this metric dropped significantly on the negative classes with $boost > 1$, especially on the `_silence_` class. In this case, even if the best beam was an empty string, the list of all the other beams contained

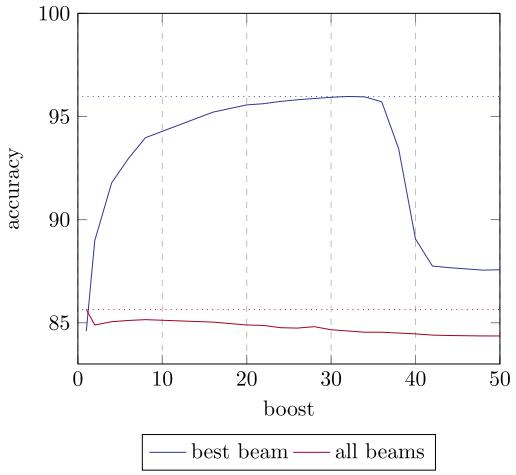

FIGURE 4. Evaluation results with unigram LM, using hypothesis from the best beam compared with hypotheses from all beams, for GSC testset.

non-empty strings. For many test cases, after boosting, one of those additional beams was equal to one of the positive classes and was counted as 0.

2) NEIGHBOR TOKENS WEIGHT BOOSTING IMPACT

After estimating the utility of the "all beams" method, we moved to "neighbor boosting". This proposition introduced one more hyperparameter to the LM. That is the reason we chose to concentrate only on the combined utilization of "all beams" and "neighbor boosting" in fine granularity, i.e., for each $n\_boost \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, we checked integer values of boost. Figure 5 shows the selected results of those evaluations on en_LS_clean. Increasing *n_boost* required larger boost values to obtain minimal EER. However, with moderate *n_boost* values, the EER dropped slightly. This is visible in Figure 5 with $n\_boost \in \{4, 8\}$. We decided that $n\_boost \in \{4, 5, 6\}$ gave the best improvement in EER compared to the necessary boost increase.

We performed similar experiments with the "best beam" and "neighbor boosting" on MOCKS but with only selected pairs of *n_boost* and *boost*. Those experiments confirmed previous observations: increasing *n_boost* requires a larger *boost* and lowers EER. However, since EER was still higher than in any of the "all beams" experiments, we omit results for "neighbor boosting" with only one hypothesis generated by the AM.

Boosting neighbor tokens as described in Section III-B2 caused a drop in the accuracy on GSC, especially with higher values of *n_boost*. The problem was once more visible in the `_silence_` class. Increasing scores of the additional tokens promoted non-empty hypotheses in the case of recordings containing non-speech data. Some of those hallucinations were equal to actual keywords from the positive classes. To remedy this, we extended the neighbor tokens set with the `<unk>` special token. It is used by the AM to represent non-speech events and re-scoring it with *n_boost* fixed the accuracy on `_silence_` class. Table 6 contains evaluation results for the "neighbor boosting" method described above.

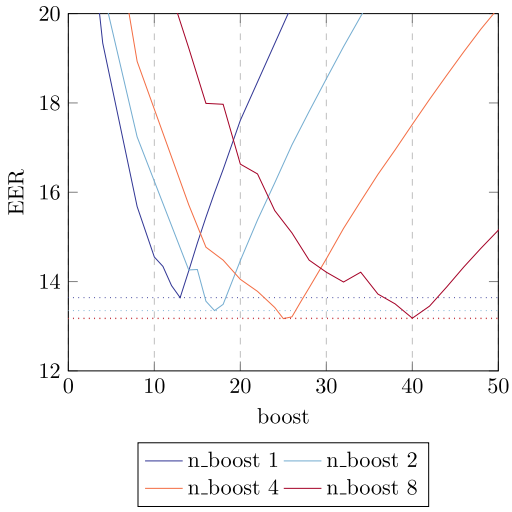

FIGURE 5. Evaluation results with unigram LM, all beams, and boosting neighbor tokens, for en_LS_clean testset.

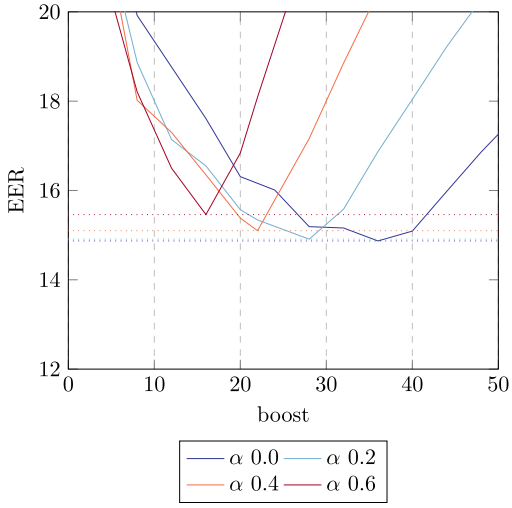

FIGURE 6. Evaluation results with unigram LM, best beam, and output smoothing, for en_LS_clean testset.

Adding "neighbor boosting" to "best beam" improved the accuracy only slightly. GSC contains simple and short phrases; thus, the impact of this feature is negligible. Similar to the case of MOCKS, in experiments on GSC, increasing *n_boost* required applying an even larger *boost*. Mixing "neighbor boosting" with "all beams" improved the accuracy from 85.64 % to 93.46 %, but this value was still far from the best result obtained with "best beam".

We also performed experiments on MOCKS with re-scoring `<unk>` token by *n_boost*, but the results were the same as without this change.

3) OUTPUT SMOOTHING IMPACT

The last modification to the baseline method we tested was smoothing the output weights. This method was applied just before LM boosting. Figure 6 shows evaluation results with output smoothing, only the best beam, and no neighbor

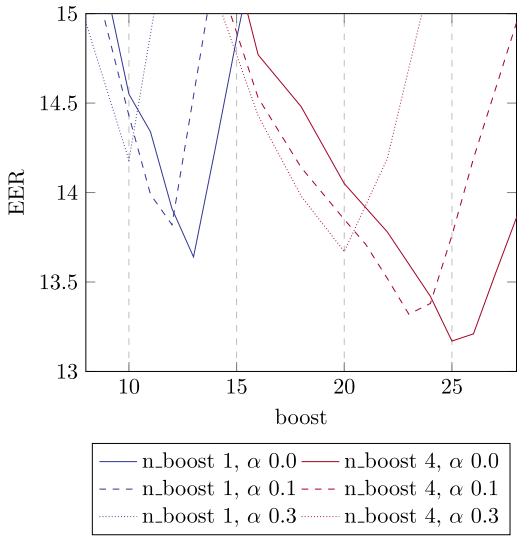

FIGURE 7. Evaluation results with unigram LM, all beams, boosting neighbor tokens, and output smoothing, for en_LS_clean testset.

boosting for en_LS_clean. Note that $\alpha = 0.0$ means no output smoothing (baseline case). We observed that with small α values, it was possible to reduce the *boost* value, which resulted in the smallest EER. With significant smoothing factors ($\alpha > 0.2$), the boost necessary to minimize EER was even lower, but, simultaneously, the model's performance deteriorated. We performed experiments with different values of α and other combinations of the proposed modifications. Figure 7 shows a closer inspection of the results obtained with ''all beams'', ''neighbor boosting'' and ''output smoothing'' for en_LS_clean. Those experiments revealed similar model behavior with different values of α: increasing the smoothing factor lowered the boost necessary to minimize EER. However, Figure 7 also shows that even with α as small as 0.1, EER increases slightly. The EER values presented in Tables 3, 4, and 5 prove that this conclusion holds for other MOCKS subsets as well.

Output smoothing with ''best beam'' applied to GSC led to similar conclusions as with MOCKS: small values of α allowed for boost reduction; however, this was also at the cost of lower accuracy. On the other hand, output smoothing applied to ''all beams'' increased the accuracy by 4% relative. Nevertheless, this value was still much lower than the baseline model with the ''best beam'' method.

C. COMPARISON OF OUR SOLUTION WITH OTHER MODELS

In Table 7, we present a comparison of the results obtained by our solution in the context of previously developed models, evaluated on LibriPhrase [37]. This testset is based on the data extracted from the training splits of LibriSpeech. It consists of two subsets: LibriPhrase Hard (LP_H) and LibriPhrase Easy (LP_E), containing phrases phonetically similar and different from the given anchor phrase. LibriPhrase became a fairly popular means of model comparison in the open vocabulary KWS community. However, we prefer evaluation

TABLE 7. Comparison of our solution tested on LibriPhrase.

model	model size	EER [%] LP_H	EER [%] LP_E
Triplet [54]	N/A	44.36	32.75
Attention [16]	582k	41.95	28.74
DONUT [23]	4M	44.36	32.75
CMCD [37]	N/A	32.90	8.42
EMKWS [38]	3.7M	23.36	7.36
baseline [11]	3.1M	37.9	5.36
ours	3.1M	23.21	3.77
CED [40]	3.8M+0.83M	14.40	1.70
AdaKWS-Tiny [41]	15M	13.47	1.61
AdaKWS-Small [41]	109M	11.48	1.21

TABLE 8. Summary of the results presented in this paper, models evaluated on MOCKS.

test set	model	EER [%]	gain
en_LS_clean	baseline [11]	14.87 ± 0.06	
	ours	13.17 ± 0.09	1.7
en_LS_other	baseline [11]	20.17 ± 0.11	
	ours	17.85 ± 0.10	2.32
en_MCV	baseline [11]	13.96 ± 0.08	
	ours	12.36 ± 0.06	1.6

on MOCKS since it is not based on the training splits of a popular dataset such as LibriSpeech. The AM used in our solution was trained, among others, on LibriSpeech training data. Thus, it might be slightly biased when evaluated on LibriPhrase. Nevertheless, we are confident that such a comparison provides a fair perspective of our solution and others.

Table 7 presents the results for EER metric and model size (where this value was available). Even though our solution does not present the best results in terms of EER, the model size is the smallest among the most recent reports. This feature is crucial with on-device keyword-spotting applications. Notably, the solution described in this paper applies to all models based on per-frame probability distribution and algorithms such as beam search. Thus, it was important to present the improvement over the baseline model. Eventually, such an improvement is visible for both MOCKS and LibriPhrase testsets.

V. DISCUSSION AND CONCLUSION

The experiments performed with MOCKS suggest that the most significant improvement in EER was gained by introducing multiple hypotheses generated by beam search into the final keyword comparator. However, it should be noted that with GSC, this modification reduced accuracy significantly. Detailed inspection of the results showed that the problem was in the test cases containing non-speech data. This issue was visible with higher boost values. Thus, it is advisable to use an additional voice activity detector (VAD, e.g. [55]) combined with an end-of-sentence detector (EOS, e.g. [56]) before the KWS solution based on multiple hypotheses and unigram LM. This way, the AM will operate on speech data only.

The goal of neighbor token boosting is to reduce the false positive rate (FPR) by improving AM performance on phrases similar to the given keyword. This improvement should also lower EER. Experiments performed on MOCKS confirmed this hypothesis. As a side-effect, we observed an increase in the boost necessary to obtain minimal EER. This behavior was expected since increasing the probability of neighbor tokens being returned by the model increased the false negative rate (FNR). To countermeasure this issue, an increase in boost was necessary. Finally, minor improvements with neighbor boosting on GSC suggest that this method works well with more complicated phrases and challenging negative samples. GSC contains elementary keywords, which can be split into one or two tokens. Furthermore, the negative samples are significantly different phonetically from the positive keywords.

In this work, we also proposed a method that reduced the most significant AM score by a fraction that was evenly distributed among all the other tokens. We tested this method jointly with LM boosting. It allowed for a decreased boost value necessary for EER minimization; however, the minimal value of this metric increased. Those effects were even more visible with larger α values. Thus, $\alpha = 0.1$ seems to be the largest value that should be used with this method.

A summary of the experiments described in this paper can be found in Table 8. It contains the EER values for the baseline model compared with the best results as shown above.

VI. FUTURE WORK

In the future, we plan to work on incorporating the total confidence scores from beam paths into a keyword comparator. Such scores could be used to weight the keyword detection decision or to reject paths with low probability. This modification could solve the issue of non-speech data recognized as a legitimate keyword.

Another interesting research topic is improving the method for neighbor token selection. The heuristic described in this paper is very simple and does not consider the phonetic similarity between tokens. This way, some phonetically similar phrases might be omitted. Furthermore, this heuristic might promote non-existent phrases, especially those that are not pronounceable. An improved heuristic for neighbor token selection should fix both issues and lower FPR and FNR.

Finally, output smoothing could be implemented with a different type of function used for score modification. The function proposed in this paper increased all the scores equally except for the largest one. Applying non-uniform smoothing might allow for the reduction of boost without a negative impact on EER.

REFERENCES

[1] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schlüter, and H. Ney, "A comprehensive study of deep bidirectional LSTM RNNS for acoustic modeling in speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 2462–2466.

[2] S. Chen, Y. Wu, Z. Chen, J. Wu, T. Yoshioka, S. Liu, J. Li, and X. Yu, "Ultra fast speech separation model with teacher student learning," in *Proc. Interspeech*, Aug. 2021, pp. 3026–3030.

[3] Z. Peng, A. Budhkar, I. Tuil, J. Levy, P. Sobhani, R. Cohen, and J. Nassour, "Shrinking bigfoot: Reducing wav2vec 2.0 footprint," in *Proc. 2nd Workshop Simple Efficient Natural Lang. Process.*, N. S. Moosavi, I. Gurevych, A. Fan, T. Wolf, Y. Hou, A. Marasović, and S. Ravi, Eds., 2021, pp. 134–141.

[4] H.-J. Chang, S.-W. Yang, and H.-Y. Lee, "Distilhubert: Speech representation learning by layer-wise distillation of hidden-unit BERT," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 7087–7091.

[5] R. Wang, Q. Bai, J. Ao, L. Zhou, Z. Xiong, Z. Wei, Y. Zhang, T. Ko, and H. Li, "LightHuBERT: Lightweight and configurable speech representation learning with once-for-all hidden-unit BERT," in *Proc. Interspeech*, Sep. 2022, pp. 1686–1690.

[6] H. D. Nguyen, A. Alexandridis, and A. Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," in *Proc. Interspeech*, Oct. 2020, pp. 3366–3370.

[7] K. Tan and D. Wang, "Towards model compression for deep learning based speech enhancement," *IEEE/ACM Trans. Audio, Speech, Language, Process.*, vol. 29, pp. 1785–1794, May 2021.

[8] E. Cohen, H. V. Habi, and A. Netzer, "Towards fully quantized neural networks for speech enhancement," in *Proc. INTERSPEECH*, Aug. 2023, pp. 181–185.

[9] O. Rybakov, P. Meadowlark, S. Ding, D. Qiu, J. Li, D. Rim, and Y. He, "2-bit conformer quantization for automatic speech recognition," in *Proc. INTERSPEECH*, Aug. 2023, pp. 4908–4912.

[10] Y. Gu, P. G. Shivakumar, J. Kolehmainen, A. Gandhe, A. Rastrow, and I. Bulyko, "Scaling laws for discriminative speech recognition rescoring models," in *Proc. INTERSPEECH*, Aug. 2023, pp. 471–475.

[11] M. Pudo, M. Wosik, and A. Janicki, "Open vocabulary keyword spotting with small-footprint ASR-based architecture and language models," in *Proc. Ann. Comput. Sci. Inf. Syst.*, Sep. 2023, pp. 657–666.

[12] R. Prabhavalkar, T. Hori, T. N. Sainath, R. Schlüter, and S. Watanabe, "End-to-end speech recognition: A survey," *IEEE/ACM Trans. Audio, Speech, Language, Process.*, vol. 32, pp. 325–351, Oct. 2024.

[13] P. M. Reuter, C. Rollwage, and B. T. Meyer, "Multilingual query-by-example keyword spotting with metric learning and phoneme-to-embedding mapping," 2023, *arXiv:2304.09585*.

[14] R. Kirandevraj, V. K. Kurmi, V. Namboodiri, and C. V. Jawahar, "Generalized keyword spotting using ASR embeddings," in *Proc. Interspeech*, Sep. 2022, pp. 126–130.

[15] 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*, Sep. 2022, pp. 5200–5204.

[16] 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 Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 6858–6862.

[17] S. Settle, K. Levin, H. Kamper, and K. Livescu, "Query-by-example search with discriminative neural acoustic word embeddings," in *Proc. Interspeech*, Aug. 2017, pp. 2874–2878.

[18] G. Chen, C. Parada, and T. N. Sainath, "Query-by-example keyword spotting using long short-term memory networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 5236–5240.

[19] A. Awasthi, K. Kilgour, and H. Rom, "Teaching keyword spotters to spot new keywords with limited examples," in *Proc. Interspeech*, Aug. 2021, pp. 4254–4258.

[20] M. Mazumder, C. Banbury, J. Meyer, P. Warden, and V. J. Reddi, "Few-shot keyword spotting in any language," in *Proc. Interspeech*, Aug. 2021, pp. 4214–4218.

[21] D. Lee, M. Kim, S. H. Mun, M. H. Han, and N. S. Kim, "Fully unsupervised training of few-shot keyword spotting," 2022, *arXiv:2210.02732*.

[22] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. 23rd Int. Conf. Mach. Learn. ICML*, 2006, pp. 369–376.

[23] L. Lugosch, S. Myer, and V. Singh Tomar, "DONUT: CTC-based query-by-example keyword spotting," 2018, *arXiv:1811.10736*.

[24] B. Kim, M. Lee, J. Lee, Y. Kim, and K. Hwang, "Query-by-example on-device keyword spotting," in *Proc. IEEE Autom. Speech Recognit. Understand. Workshop (ASRU)*, Dec. 2019, pp. 532–538.

[25] 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.

[26] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden Markov modeling for speaker-independent word spotting," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, vol. 1, 1989, pp. 627–630.

[27] J. G. Wilpon, L. G. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, vol. 1, 1991, pp. 309–312.

[28] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.

[29] I.-F. Chen and C.-H. Lee, "A hybrid HMM/DNN approach to keyword spotting of short words," in *Proc. Interspeech*, Aug. 2013, pp. 1574–1578.

[30] 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*, Sep. 2016, pp. 760–764.

[31] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1–9.

[32] K. Hwang, M. Lee, and W. Sung, "Online keyword spotting with a character-level recurrent neural network," 2015, *arXiv:1512.08903*.

[33] Y. Zhuang, X. Chang, Y. Qian, and K. Yu, "Unrestricted vocabulary keyword spotting using LSTM-CTC," in *Proc. Interspeech*, Sep. 2016, pp. 938–942.

[34] S. Sigtia, P. Clark, R. Haynes, H. Richards, and J. Bridle, "Multi-task learning for voice trigger detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 7449–7453.

[35] T. Bluche and T. Gisselbrecht, "Predicting detection filters for small footprint open-vocabulary keyword spotting," in *Proc. Interspeech*, Oct. 2020, pp. 2552–2556.

[36] 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 Autom. Speech Recognit. Understand. Workshop (ASRU)*, Dec. 2017, pp. 474–481.

[37] H.-K. Shin, H. Han, D. Kim, S.-W. Chung, and H.-G. Kang, "Learning audio-text agreement for open-vocabulary keyword spotting," in *Proc. Interspeech*, Sep. 2022, pp. 1871–1875.

[38] K. Nishu, M. Cho, and D. Naik, "Matching latent encoding for audio-text based keyword spotting," 2023, *arXiv:2306.05245*.

[39] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," in *Proc. Interspeech*, Oct. 2020, pp. 5036–5040.

[40] K. Nishu, M. Cho, P. Dixon, and D. Naik, "Flexible keyword spotting based on homogeneous audio-text embedding," 2023, *arXiv:2308.06472*.

[41] A. Navon, A. Shamsian, N. Glazer, G. Hetz, and J. Keshet, "Open-vocabulary keyword-spotting with adaptive instance normalization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2024, pp. 11656–11660.

[42] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022, *arXiv:2212.04356*.

[43] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*.

[44] C.-C. Chiu and C. Raffel, "Monotonic chunkwise attention," 2017, *arXiv:1712.05382*.

[45] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[46] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," 2015, *arXiv:1508.07909*.

[47] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 5206–5210.

[48] 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 *Proc. Int. Conf. Lang. Resour. Eval.*, 2019, pp. 4218–4222.

[49] 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 Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 776–780.

[50] 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*, Sep. 2019, pp. 2613–2617.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015, *arXiv:1512.00567*.

[52] M. Pudo, M. Wosik, J. Krzywdziak, B. Lukasiak, and A. Janicki, "MOCKS 1.0: Multilingual open custom keyword spotting testset," in *Proc. INTERSPEECH*, Aug. 2023, pp. 4054–4058.

[53] B. Sanchez-Lengeling, J. N. Wei, B. K. Lee, R. C. Gerkin, A. Aspuru-Guzik, and A. B. Wiltschko, "Machine learning for scent: Learning generalizable perceptual representations of small molecules," 2019, *arXiv:1910.10685*.

[54] N. Sacchi, A. Nanchen, M. Jaggi, and M. Cernak, "Open-vocabulary keyword spotting with audio and text embeddings," in *Proc. Interspeech*, 2019, pp. 3362–3366.

[55] T. Hughes and K. Mierle, "Recurrent neural networks for voice activity detection," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, May 2013, pp. 7378–7382.

[56] M. Pudo, A. Wiśniewski, and A. Janicki, "Improved weighted loss function for training end-of-speech detection models," in *Proc. 18th Int. Conf. Adv. Mobile Comput. Multimedia*, Chiang Mai, Thailand, Nov. 2020, pp. 25–29.

MIKOŁAJ PUDO received the M.Sc. degree in computer science from Jagiellonian University, Kraków, Poland, in 2006. He is currently pursuing the Ph.D. degree with the Warsaw University of Technology. He is with the Samsung Research and Development Institute Poland, Kraków. His research interests include automatic speech recognition, keyword detection, speech enhancement, and optimization of speech processing methods toward on-device applications.

MATEUSZ WOSIK received the M.Sc. degree in computer science from the AGH University of Science and Technology, Kraków, Poland, in 2018.

Since 2019, he has been an NLP Engineer with the Samsung Research and Development Institute Poland, Kraków. His research interests include natural language processing, with a focus on speech-processing technologies.

ARTUR JANICKI (Member, IEEE) received the M.Sc. (Hons.), Ph.D. (Hons.), and D.Sc. (Habilitation) degrees in telecommunications from the Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT), Warsaw, Poland, in 1997, 2004, and 2017, respectively. He is currently an Associate Professor with the Cybersecurity Division, Institute of Telecommunications, WUT. He is the author or co-author of over 90 conferences and journal articles and the supervisor of over 80 B.Sc. and M.Sc. theses. His research and teaching activities focus on signal processing and machine learning, mostly in the cybersecurity context. He is a member of ISCA. He is also a member of the technical program committees of international conferences and a reviewer of international journals in computer science and telecommunications.