Warsaw University of Technology

Discipline of Science Engineering and technology
Field of Science Information and Communication Technology

Ph.D. Thesis

Mikołaj Pudo, M.Sc.

Methods of optimizing models and algorithms for automatic speech recognition in mobile applications

(Metody optymalizacji modeli i algorytmów automatycznego rozpoznawania mowy pod kątem działania na urządzeniach mobilnych)

Supervisor
Prof. Artur Janicki, Ph.D., D.Sc.

Warsaw 2023

Metody optymalizacji modeli i algorytmów automatycznego rozpoznawania mowy pod kątem działania na urządzeniach mobilnych

Streszczenie. Interfejsy głosowe stają się coraz bardziej popularnym sposobem komunikacji użytkownika z urządzeniami elektronicznymi. Obecnie są one już standardem w aplikacjach typu inteligentny asystent. Dane zawierające komunikaty głosowe charakteryzują się znaczną heterogenicznością. Źródła tej różnorodności mogą wystąpić już na poziomie rejestrowanego sygnału audio. Na jakość tego sygnału wpływ mają czynniki takie jak: typ mikrofonu rejestrującego dźwięk, szum tła, sposób artykulacji mówcy. Różnorodność może pojawić się również na poziomie językowym, ponieważ ta sama intencja może być wyrażona na wiele sposobów w tym samym języku. Ponadto liczba sposobów wyrażania jednej intencji znacząco się zwiększa przy rozważaniu wielu języków. Proces przetwarzania języka naturalnego przeważnie jest rozłożony na wiele etapów. W niniejszej rozprawie zaprezentowane są wyniki badań dotyczących pierwszego etapu tego procesu: konwersji mowy zawartej w strumieniu audio do zapisu tekstowego. W szczególności badania dotyczyły następujących modułów wykorzystywanych w tym procesie: wykrywanie fraz kluczowych w strumieniu audio, wykrywanie końca komendy, czy wreszcie samo dekodowanie strumienia audio do tekstu. Ponadto badania były ukierunkowane na opracowanie metod umożliwiających przeniesienie możliwie wielu wspomnianych modułów na urządzenia mobilne. Może się to odbyć poprzez opracowanie nowatorskich algorytmów lub modeli, ale również optymalizację działania istniejących rozwiązań.

W przypadku zadania wykrywania fraz kluczowych problematyczne okazało się już określenie procedury testowania tego typu rozwiązań. Dotychczasowe prace opierały się na wynikach ewaluacji przeprowadzanych na zbiorach danych, które zawierają bardzo mało fraz kluczowych, nie zawierają wymagających przykładów negatywnych lub nie są dostępne publicznie. W ramach pierwszego etapu prac przygotowano zbiór danych testowych rozwiązujący powyższe problemy i udostępniono go publicznie. Zbiór ten został nazwany Multilingual Open Custom Keyword Spotting Testset (MOCKS). Jest on oparty na publicznie dostępnych bazach audio: LibriSpeech i Mozilla Common Voice. MOCKS zawiera prawie 50 000 fraz kluczowych dla pięciu języków: angielskiego, francuskiego, hiszpańskiego, niemieckiego i włoskiego.

Kolejnym etapem badań było opracowanie metody poprawiającej skuteczność działania systemu wykrywania fraz kluczowych opartego na współczesnych modelach akustycznych. Zaproponowana metoda polega na zastosowaniu prostego modelu językowego z odpowiednio dobranymi wagami ustalanymi w fazie inicjalizacji. Przeprowadzone eksperymenty z ewaluacjami wykonanymi na zbiorze MOCKS pokazały, że możliwe jest takie ustawienie parametrów modelu językowego, które zwiększa miarę prawdziwie pozytywną (ang. *true positive rate*) o około 30 punktów procentowych, z jednoczes-

nym wzrostem miary fałszywie negatywnej (ang. *false positive rate*) o około 20 punktów procentowych. Ponadto przeprowadzono eksperymenty z wykorzystaniem nagrań wygenerowanych przez syntezator mowy, mające na celu poprawę skuteczności modelu w przypadku specyficznych i rzadko występujących fraz kluczowych.

Efektywność systemów automatycznego rozpoznawania mowy zależy między innymi od dokładności wyznaczenia początku i końca fragmentu sygnału zawierającego mowę. W szczególności istotne i skomplikowane jest wyznaczenie momentu końca frazy wypowiadanej przez użytkownika. W ramach tego problemu badawczego zaproponowano ulepszoną metodę treningu modeli wykrywających koniec komendy w strumieniu audio. Innowacją jest tu rozszerzenie standardowej funkcji kosztu o zestaw wag przypisanych do różnych części danych audio. W trakcie badań zaproponowano również metodę doboru tych wag. Przeprowadzone eksperymenty potwierdzają skuteczność opracowanej funkcji kosztu w porównaniu ze standardowym sposobem treningu modeli.

Dekodowanie strumienia audio do tekstu jest przeważnie ostatnim etapem konwersji mowy do zapisu słownego. W trakcie treningu modele stosowane w tym zadaniu wymagają dużych ilości wysokiej jakości danych. Łatwo dostępne są obszerne bazy danych audio, natomiast proces ich anotacji przeważnie jest robiony ręcznie. Z tego powodu jest on kosztowny i czasochłonny. Metody uczenia częściowo nadzorowanego stanowią próbę rozwiązania tego problemu. We wcześniejszych pracach eksperymentalnie wykazano ich skuteczność w przypadku bardzo dużych zbiorów danych audio. Natomiast badania zaprezentowane w niniejszej rozprawie pokazują, że możliwe jest zastosowanie uczenia częściowo nadzorowanego również w przypadku niewielkich baz danych. Wykorzystując zaproponowane usprawnienia do podstawowej metody, udało się obniżyć wyrazową stopę błędów (ang. *word error rate*, WER) o 12–22 punktów procentowych (w zależności od typu zbioru audio wykorzystanego do adaptacji modelu bazowego).

Słowa kluczowe: uczenie maszynowe, głębokie sieci neuronowe, przetwarzanie mowy, przetwarzanie języka naturalnego, rozpoznawanie mowy na urządzeniu, interfejs głosowy, urządzenia mobilne, projektowanie korpusów, wykrywanie fraz kluczowych, wykrywanie końca frazy w strumieniu audio, wykrywanie aktywności głosowej w strumieniu audio, rozpoznawanie mowy, uczenie częściowo nadzorowane

Methods of optimizing models and algorithms for automatic speech recognition in mobile applications

Abstract. Voice-based interfaces are becoming an increasingly popular way of interaction between humans and machines. Nowadays, they are a standard in applications such as intelligent assistants. The data containing voice messages is characterized by considerable heterogeneity. This diversity may occur at the level of the recorded audio signal. The quality of this signal is affected by factors such as the type of microphone recording the sound, background noise, or the speaker's articulation. Diversity can also occur at the linguistic level, as the same intention can be expressed in many ways in the same language. Moreover, the number of ways to express one intention increases significantly when considering multiple languages. Due to the above observations, the natural language processing process is usually divided into many stages. This dissertation presents the research results on the first stage of this process: automatic speech recognition. In particular, the study concerned the following modules used in this process: keyword spotting, end-of-speech detection, and decoding the audio stream to text. In addition, the research was focused on developing methods that would enable the transfer of as many of these modules as possible to mobile devices. This can be done by developing innovative algorithms or models and optimizing the operation of existing solutions.

Analysis of the previous work on keyword spotting revealed an issue with the lack of a good testset that could be used to evaluate models for this task. Previous solutions were evaluated with the testsets that contain very few keywords, do not contain demanding negative examples, or are not publicly available. Therefore, in the first stage of work, a testset solving the above problems was prepared and made publicly available. This testset was named Multilingual Open Custom Keyword Spotting Testset (MOCKS). It is based on open-domain datasets: LibriSpeech and Mozilla Common Voice. MOCKS contains almost 50 000 keywords for five languages: English, French, German, Italian, and Spanish.

Once the testset was ready, the next research stage was to develop a method to improve the effectiveness of the keyword detection system based on modern acoustic models. The proposed solution uses a simple unigram language model. This thesis also presents a method for weight selection during the initialization phase. The proposed solution was evaluated with the MOCKS testset. Experiments showed that with proper choice of the language model parameters, it was possible to increase the true positive rate by about 30 % relative while increasing the false negative rate only by about 20 % relative. In addition, experiments were carried out using recordings generated by a text-to-speech system to improve the model's effectiveness with rare keywords unknown during the training phase.

The effectiveness of automatic speech recognition systems depends, among other things, on the accuracy of determining the beginning and end of a fragment containing speech. In particular, it is crucial and demanding to locate the position of the

end-of-speech event in the audio stream. This thesis describes an improved method of training models detecting such an event. The innovation extends the classic binary cross-entropy loss function with weights assigned to different parts of the audio data. A method of assigning these weights was also proposed. The experiments confirm the proposed loss function's effectiveness compared with the standard model training method.

Decoding the audio stream to text is usually the last step in speech-to-text conversion. Models used in this task typically require large amounts of high-quality data for training and adaptation. Extensive audio databases are readily available, but the annotation process is mainly done manually. For this reason, it is costly and time-consuming. Semi-supervised learning methods attempt to solve this issue. Previous work has experimentally demonstrated their effectiveness for massive audio datasets. However, the research presented in this dissertation shows that it is possible to use semi-supervised learning in the case of small datasets as well. Using the proposed improvements to the baseline method, it was possible to reduce the word error rate metric by 12 %–22 % relative (depending on the type of audio set used to adapt the base model).

Keywords: machine learning, deep neural network, speech processing, on-device speech recognition, voice interface, mobile devices, corpus design, keyword spotting, custom keyword spotting, query-by-example keyword spotting, query-by-text keyword spotting, end-of-speech detection, voice activity detection, speech recognition, semi-supervised learning

Acknowledgements

I would like to express my gratitude to my wonderful wife Joanna and son Filip for their patience and support in the journey that led to writing this thesis.

I would also like to acknowledge and warmly thank my WUT supervisor, Prof. Artur Janicki, Ph.D., D.Sc., and my Samsung supervisor, Bożena Łukasiak, Ph.D.

Finally, I would like to thank all my Samsung R&D Institute Poland colleagues for many inspiring discussions and support in research activities.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

AD – Audio Decoding
AE – Acoustic Engine
AEC – Acoustic Echo Cancellation
AI – Artificial Intelligence
AM – Acoustic Model
ASR – Automatic Speech Recognition
BOS – Beginning-of-speech Detection
BiLSTM – Bidirectional Long Short-term Memory
BPE – Byte Pair Encoding
CRF – Conditional Random Field
CTC – Connectionist Temporal Classifier
DNN – Deep Neural Network
EOS – End-of-speech Detection
FNR – False Negative Rate
FPR – False Positive Rate
FST – Finite State Transducer
GSC – Google Speech Commands
GMM – Gaussian Mixture Model
HMM – Hidden Markov Model
IA – Intelligent Assistant
LLM – Large Language Model
LM – Language Model
LSTM – Long Short-term Memory

KWS – Keyword Spotting
MCV – Mozilla Common Voice
MFCC – Mel-frequency Cepstral Coefficients
ML – Machine Learning
MOCKS – Multilingual Open Custom Keyword Spotting Testset
NLG – Natural Language Generation
NLP – Natural Language Processing
NLU – Natural Language Understanding
NR – Noise Reduction
QbyE – Query-by-example
QbyT – Query-by-text
RIR – Room Impulse Response
SBD – Speech Boundaries Detection
SER – Sentence Error Rate
SSL – Semi-supervised Learning
SVM – Support Vector Machine
TNR – True Negative Rate
TPR – True Positive Rate
TTS – Text-to-speech
WER – Word Error Rate
VAD – Voice Activity Detection

1. Introduction

1.1. Intelligent Assistants and Speech Processing

We have the privilege of observing the technological revolution taking place. It is the revolution in the way we communicate with machines. The variety of tasks performed by computers has been quickly expanding since the middle of the 20th century. It might be argued that there has been a feedback loop: the more accessible communication with computers was, the more helpful they became, which stimulated inventions in machine interface technologies. In the 1970s, punched cards were replaced by a keyboard and a computer monitor. Soon, a mouse was introduced. Hence, for many years, the primary way of human-computer interaction was based on written text and pointing devices. This is still true, even though computers have changed significantly. Today's mobile devices have far stronger processors and use orders of magnitude more memory than Apple 1, the first personal computer released in 1976 [1]. Yet the communication schema is very similar in both cases: users must enter text by pressing keys on a keyboard. There were also attempts with steering machines with gestures [2], [3] or using the movement of the pupil [4], [5]. However, speaking is a very straightforward and natural way of communication for humans. This observation motivated researchers to develop new solutions enabling machines to decode speech and parse the transcription.

Speech and natural language are challenging to process since many factors affect the quality and quantity of the data:

- Background environment (acoustic echo, reverberation, background noise).
- Substantial differences between languages.
- People speak differently (rhythm, tone, accent).
- Speaking disorders.
- The same commands can be formulated with different words.

The number of applications that exploit Speech and Natural Language Processing is quickly expanding. Some of the most prominent applications in this field are:

- Intelligent Assistants (IA) [6], [7],
- Interactive Voice Response in telephony [8], [9],
- Education (language learning, visually impaired students support) [10], [11],
- Support for people with disabilities in everyday life [12], [13],
- Home automation [14], [15].

Throughout this thesis, the focus will be on Intelligent Assistants. This choice is motivated by the fact that this is a rapidly growing software branch, and multiple consumer devices support such applications (mobile phones, watches, TVs, fridges, vacuum cleaners, etc.).

Usually, many distinct components are used in the pipeline designed to convert spoken

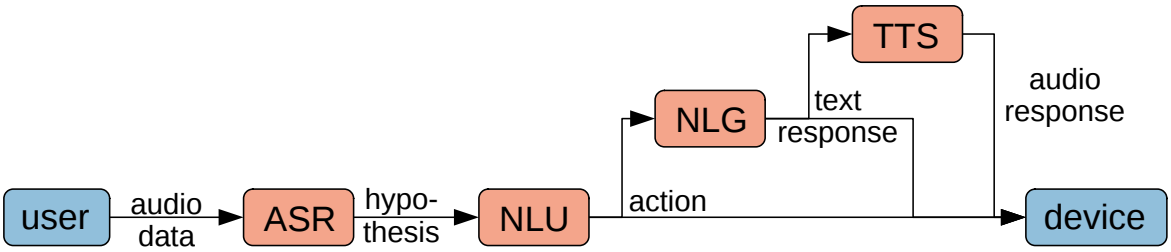

Figure 1. Overview of a generic intelligent assistant solution.

commands into machine actions. An example of such a pipeline, used commonly in the case of IAs, is shown in Figure 1.

The workflow of this pipeline can be described as follows:

1. The user provides audio data recorded by a microphone (or a microphone matrix).
2. Audio data is decoded by the Automatic Speech Recognition (ASR) module. The output of this module is a textual transcription of the command, usually called a hypothesis.
3. ASR's hypothesis is processed by the Natural Language Understanding (NLU) module. NLU's goal is to parse the command into action, which can be performed by the device.
4. One of the goals of IAs is to simulate conversation with the user. Natural Language Generation (NLG) is used to create a sentence that would be an answer to a user's request.
5. Finally, IA's textual answer is converted into audio by a Text-to-speech (TTS) module.

In this thesis, we will focus on the ASR module. This module can be further split into submodules. Figure 2 shows the most generic ASR pipeline. Each module is briefly described in subsections 1.2-1.6.

1.2. Keyword Spotting

In the early versions of IAs, each conversation would start with a dedicated button (push-to-talk solution), but soon the hands-free option was introduced. Currently, all leading IAs use keyword detection for conversation initialization. The goal is to detect all occurrences of the given keywords in audio data provided either in streaming or non-streaming mode. Keyword-spotting systems (KWS) 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 or just one keyword (e.g., "OK Google", "Alexa", "Hey Siri" or "Hi Bixby"). Such models can be minimized well, even to sizes below 100 kB. However, users of IAs often request the possibility to customize the keywords, which requires open-vocabulary KWS. In this case, the model must be much larger to recognize potentially arbitrary keywords. The problem of open-vocabulary KWS comes in two flavors:

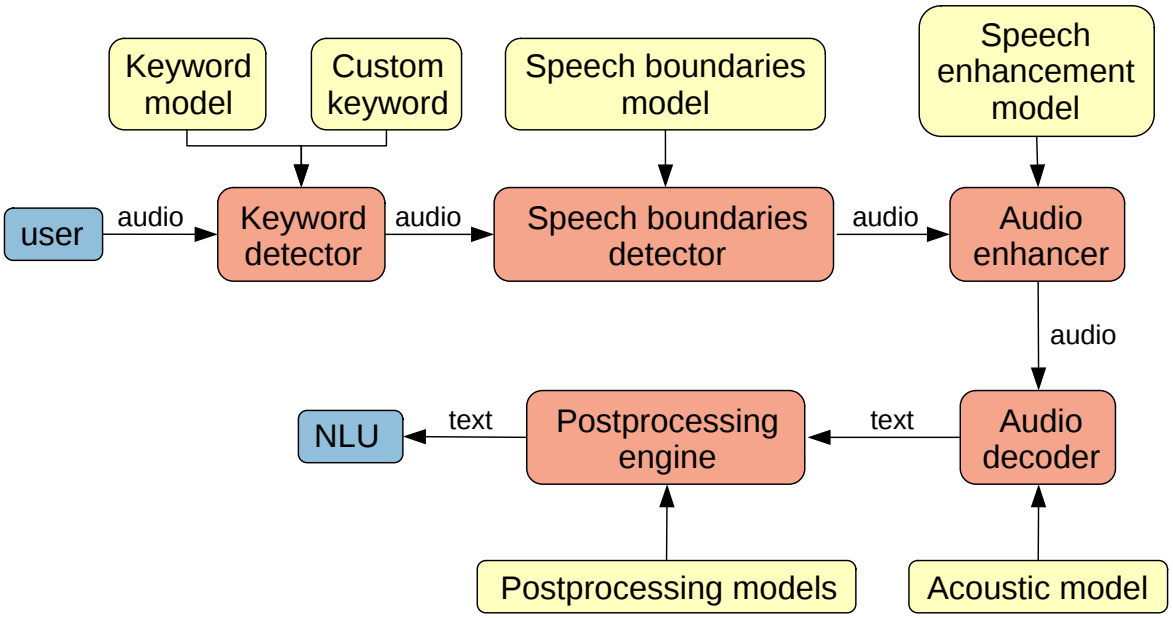

Figure 2. Overview of the generic speech recognition solution.

1. custom keyword is provided by text (in literature: *query-by-text keyword spotting* or QbyT KWS for short),
2. custom keyword is provided by audio (in literature: *query-by-example keyword spotting* or QbyE KWS for short).

In both cases, an additional enrollment phase is necessary to gather the keyword from the user and extract the features for further comparison. Throughout this thesis, KWS will mean both QbyT KWS and QbyE KWS tasks.

1.3. Speech Boundaries Detection

Once the KWS module activates the IA, the audio data is gathered for further processing. However, it is not obvious when to start and stop recording. The audio decoder's performance depends heavily on the precision of the audio stream truncation. Opening the stream too late or closing it while the command is still being spoken prevents the decoder from processing the entire utterance correctly. On the other hand, the Acoustic decoder's performance often depends on the amount of data presented to the decoder, so feeding it with a redundant signal is not advantageous. Furthermore, the user experience in a real-time application depends strongly on the system latency: only once the Audio decoder is notified that the utterance is finished can further components start processing the request. The tasks of detecting the beginning and end of speech are usually called Beginning-of-speech (BOS) and End-of-speech (EOS) Detection, respectively. They are sub-tasks of Speech Boundaries Detection (SBD).

Defining precise moments of speech boundaries is a task that needs to be performed at the very beginning of the speech recognition process. Additionally, since audio decoding

is usually performed in real-time, detecting speech boundaries, apart from being precise, should also be done quickly, in a one-time, single-run manner. This means that the models designed for this task can infer only based on the current data that have been presented, with no chance for any further corrections of past hypotheses.

1.4. Audio Enhancement

Mobile devices are used in various acoustic conditions, which results in the audio data containing background noise. Such distortions heavily affect the output of the following modules. They can cause errors during audio decoding. Those errors propagate to NLU, which will fail to parse the user's command. Various techniques can be applied to enhance the audio data. Some of those techniques can be as follows:

- Acoustic Echo Cancellation (AEC) [16]. Many consumer devices are equipped with loudspeakers, e.g., mobile phones, TV sets, car audio systems). Microphones also record sounds generated by those loudspeakers. However, the device's audio signal can be subtracted from the recorded signal by the AEC module.
- Noise Reduction (NR) [17] removes noise from the audio signal. It can be split into two subtasks: stationary and non-stationary noise reduction. Usually, the former is easier, while the latter is much harder and requires sophisticated models.
- Speech Amplification has a goal opposite to NR: it is meant to amplify frequencies containing speech data.

1.5. Audio Decoding

Audio Decoding (AD) [18] can be considered the core of ASR since its goal is to generate a transcription of the command spoken by the user. Usually, this task is performed by the acoustic engine (AE) and acoustic model (AM). Multiple factors affect the acoustic model's performance. The most important are vocabulary, pronunciation variants, context, and acoustic conditions.

With the most simple use cases of AD, it can be assumed that the vocabulary is closed and the context is small. It can also be assumed that the acoustic conditions in the production environment will be constant. Such assumptions can be made, e.g., for elementary sets of supported commands or devices used only in one location. In this case, the AM can be relatively small. Nevertheless, the high level of generality in use cases supported by the AM requires a large number of parameters in the model and, consequently, significant computing power.

A language model (LM) can optionally process the output from AE. In this case, the goal of LM is to capture the most common patterns in a given language and apply this knowledge to choose the most probable hypothesis generated by the AM.

1.6. Postprocessing

The last module of ASR is usually postprocessing. However, this step can be optional. This module is different from the previous ones since it is purely text-based. There can be multiple tasks performed within this step. Some examples are as follows:

- Inverse text normalization [19]. Users can pronounce phrases such as numbers, dates, weights, distances, or addresses in numerous ways. However, they can be written canonically, which might be language-dependent. Different methods can perform such normalization, among which simple grammar can be considered a natural candidate.
- Fixing the most common audio decoder errors. The AD usually uses an extensive neural network to produce the hypothesis. Fine-tuning such models might not be straightforward, and it might be time-consuming. Once an error in the AD output is spotted, applying a text-based method to fix it might be much faster and cheaper. Such methods include simple regular expression rules, grammar, or text processing models.

1.7. Client-server vs. On-device Processing

Contemporary models used in the ASR process are usually large and require massive computing power. In previous years, such power was available only on expensive servers. Hence, the dominant paradigm in ASR was to apply client-server architecture. Some lightweight tasks were performed on-device, yet the most compute- and memory-consuming tasks were delegated to the cloud. In general, tasks such as KWS and SBD were performed on the device since their output directly impacted the hardware (the microphone). Audio Enhancement should be partially performed on the device (AEC), but a fraction of this task can also be performed in the cloud. Audio Decoding and Postprocessing were, in general, performed in the cloud.

New versions of consumer devices are equipped with strong processors and large amounts of memory. Simultaneously, researchers improve model architectures and training methods. This grants the possibility to move more ASR components to mobile devices, particularly the most computationally expensive one: Acoustic Decoding. However, we are only at the beginning of this research.

1.8. Scope and Organization of this Thesis

It should be noted that, as outlined before, ASR is a complex process consisting of multiple submodules. My work in this thesis concentrates on the below three modules:

- Keyword detector,
- Speech boundaries detector and End-of-speech detector in particular,

- Audio decoder and acoustic model adaptation with Semi-supervised Learning methods in particular.

The rest of this thesis is organized as follows. Section 2 presents previous work in the areas related to the abovementioned modules. In Section 3, the main theses of my doctoral dissertation are stated and shortly introduced. Detailed work on those theses is presented in Sections 4, 5, and 6. Conclusions are presented in Section 7. Finally, the author's achievements are listed in Appendix A.

2. Literature Review

In this section, I will present a survey of the previous works on the topics investigated in this thesis.

2.1. Custom Keyword Spotting

KWS can be split into 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 [20]. Models designed for this task evolved similarly to the acoustic models (AM) used in speech recognition. There was a long phase of solutions based on the hidden Markov model (HMM) – Gaussian mixture model (GMM) architecture [21], [22] (Figure 3a). In this approach, the acoustic features were modeled by the GMM. The output of the GMM was used as the HMM emission probabilities. Finally, Viterbi decoding [23] 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) [24], [25]. Finally, with the advent of end-to-end architectures to speech processing, ideas such as connectionist temporal classification (CTC) [26] and attention mechanism [27] became standard solutions in KWS as well. In this approach, the model was usually composed of the encoder and decoder parts (Figure 3b-d). The goal of

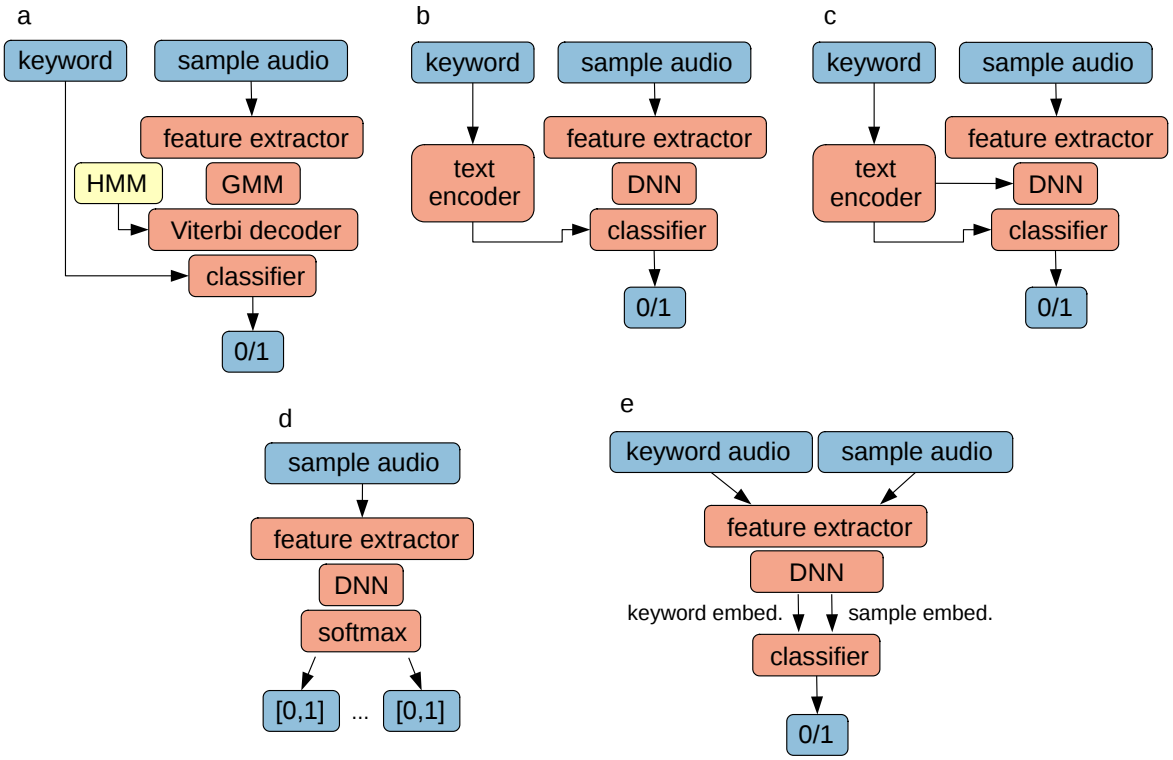

Figure 3. Overview of the most common KWS model architectures.

the encoder was to generate frame-level embeddings. The decoder was trained to compile those embeddings into a sequence of posterior probabilities, usually much sparser than the frame sequence. Proposed solutions differ in how the posteriors are generated by the decoder and handled in the postprocessing step. Another difference between solutions is in the keyword encoder module. In some solutions, this was simply an identity function or translation to phonetic transcription, while in others, a more sophisticated transformation was applied.

Contemporary solutions to open-vocabulary KWS can be based simply on CTC. In [28], the model architecture was similar to the schema presented in Figure 3b. The encoder part of the model contained three long short-term memory (LSTM) layers and a softmax output layer generating character-level probabilities. The keyword was detected once the negative log posterior was below a predefined threshold. LSTM-CTC architecture was also used in [29] (Figure 3b). The solution was designed to detect multiple keywords simultaneously. In this case, the model operated at the phonetic level. One or more phone sequences represented the keyword. Those variants were compared with the hypothesis using minimum edit distance during the inference phase. Each keyword's decision threshold was estimated separately based on the training data and the lexicon.

The connection between KWS and acoustic modeling can also be limited to the training phase. In [30], the model was trained using CTC in a multi-task approach with speech recognition and KWS outputs. During inference, only the KWS output was used. Such an approach was intended to improve the model's generalization ability and performance in acoustically challenging conditions. However, the number of recognized keywords was fixed (Figure 3d), and extending the model to support new keywords required retraining. Hence, its utility for open-vocabulary KWS was limited. The solution presented in [31] was based on a keyword spotting network and a text keyword encoder (Figure 3b). The keyword spotting network had two components: an audio encoder network and a convolutional classifier. The audio encoder was trained using the speech recognition task. The classifier network used filters computed by a keyword encoder. The keyword encoder was a bidirectional LSTM (BiLSTM) layer processing the text keyword provided by the user.

Some solutions were based on the attention mechanism in the decoder part of the model. In [32], three types of encoder-decoder models were proposed. In the best-performing architecture, the decoder already used the keyword embedding (Figure 3c). The idea was to employ such embedding in the attention layer to bias the decoder toward the keyword and, in effect, improve recognition of the searched phrase. Notably, one of the models presented in this paper employed a phoneme level 6-gram LM (~1.5 M 6-grams), which improved the model's performance. An attention-based model was presented in [33]. However, this was one of many solutions with a fixed-size output layer (Figure 3d); hence, supporting new keywords required retraining the model.

Another approach to supporting large vocabulary was the on-the-fly adaptation of the

model during the initialization phase. In [34], an embedding model was pre-trained on a significant but limited number of classes (Figure 3d). A classification layer for specific keywords was added on top of the embedding model and adapted using only a handful of samples. Such classification layers were independent of each other and used the same embedding model since only the last layer was modified in the adaptation phase. A similar solution based on a few-shot transfer learning was described in [35] (Figure 3d). 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.

The goal of QbyE KWS is to compare two audio samples. Hence, the most generic approach to this task is to generate two embeddings with the same DNN and compare them (Figure 3e). Many QbyE solutions were also based on the concepts used in speech recognition. In [36], the AM with CTC was applied to enrollment phase recordings. N-best phonetic level keyword labels were stored with their log probabilities in the keyword model. Each audio was processed with a similar AM during the inference phase. Log probability was computed and added to the final score for each keyword from the keyword model. The keyword was detected if the score was above a certain pre-determined threshold. Similarly, in [37], a small-footprint AM based on CTC was employed. In the enrollment phase, phonetic level posteriorgrams obtained from the model were used to build a finite-state transducer graph (FST) that modeled the keywords. 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. Since both solutions operated on the phonetic level rather than directly on the acoustic level, they can be treated as converting the QbyE task to QbyT.

QbyE can also be approached on the acoustic level. In [38]–[41], audio embeddings were computed for both enrollment and inference phase recordings. Distance between those vectors was calculated using different metrics and compared to a predefined threshold.

It should be noted that contrary to the speech recognition task, in KWS, there is no good public testset that could be used to evaluate models, especially for custom KWS scenarios. Table 1 contains a list of the most popular testsets. They are also described below in detail.

Many KWS solutions were tested using the Google Speech Commands (GSC) [42]. It was released in two versions, V1 and V2, containing recordings of 30 and 35 words, respectively. Ten words were treated as positive samples (keyword), and the remaining part was used as negative samples (non-keyword). The words were short enough to fit in the recording, lasting under one second. The GSC testset contained crowdsourced recordings from 1 881 speakers (V1) and 2 618 speakers (V2). Several studies reported their results evaluated on V1 [33], [43]–[45], while the others on V2 [33], [43], [46], [47]. The

Table 1. Testsets used to evaluate KWS solutions.

testset name	**publicly available**	**pros**	**cons**
Google Speech Commands [42]	✓	– many speakers, – many samples from each speaker.	– small number of phrases, – phrases are very different, – clean audio.
Multilingual Spoken Words Corpus [48]	✓	– massively multilingual, – many phrases.	– single word phrases, – no negative test cases.
Mozilla Common Voice [49]	✓	– many speakers, – many samples from each speaker.	– small number of phrases, – phrases are very different.
LibriSpeech based [36]	✗	– varied keywords, – large number of keywords, – negative test cases.	– contains short phrases, – not available publicly.
LibriSpeech based [50]	✗	– large number of keywords, – multiple subsets with different size.	– contains very short phrases, – not available publicly.
MCV based [34]	✗	– large number of keywords, – large number of test cases per keyword.	– no negative test cases, – not available publicly.

best results so far have been reported by [43] with an accuracy of 98.0 % on V1 and 98.7 % on V2.

Despite its popularity, the GSC testset could not evaluate KWS solutions thoroughly for several reasons: the number of keywords was relatively low, and the negative samples were entirely different from the keywords. All the samples were recorded without background noise and were cut with high precision. Those issues make GSC inadequate for emulating actual production conditions.

The Multilingual Spoken Words Corpus (MSWC) [48] contains as many as 23 M keywords split between 50 languages. Alignments generated with Montreal Forced Aligner (MFA) [51] were used to extract audio samples. The keywords were selected based on the word length (minimum three letters) and occurrence frequency (minimum five occurrences per language subset). The data was split between *train*, *dev*, and *test* subsets. Unfortunately, this dataset does not contain keywords longer than one word. Moreover, it does not provide negative samples to a given keyword, which would allow for a false positive rate (FPR) analysis.

Mozilla Common Voice (MCV) [49] contains subsets of short phrases called “Single Word Target Segment”, which could be used for KWS evaluation. This approach was applied in [34]. However, those subsets are also very limited since they contain only up

to 14 short keywords (digits and four predefined keywords). Furthermore, there are no negative samples defined for each keyword.

A few papers used modifications of datasets available in the public domain to evaluate proposed solutions. One example is [36], where LibriSpeech [52] was the base for creating the testset. Alignments generated with MFA were used to extract audio samples. The testset contained recordings 0.5 s–1.5 s long, including n-grams with $n \leq 4$. This gave 6 047 different keywords. Positive samples were combined with two types of negative samples: "confusing" (phonetically similar to the keyword) and "non-confusing" (phonetically dissimilar). Phoneme-level transcriptions and edit distances were used to select both types of negative samples. The description of the testset was fairly precise, but the testset itself has not been published.

In [50], a KWS solution based on triplet loss was evaluated; apart from GSC, the test suite included recordings extracted from LibriSpeech. Different testsets were created using 10, 100, 1 000, and 10 000 most popular words. Forced alignments were used to extract selected phrases. Since the most popular words were "the", "and", "of", etc., the audio excerpts were relatively short (0.03 s–2.8 s). Unfortunately, no information about the sizes of the classes in the testset was given, nor has the testset been published.

The GSC testset, excerpts from LibriSpeech based on alignments generated by MFA, and subsets of MCV were used in [34]. In this case, the testset was built with n-grams ($n \leq 5$) that contained at least ten characters and had at least ten occurrences in the train split of the dataset. This gave over 15.2 k different keywords. Unfortunately, this testset has not been made public, either.

In [53], a procedure for creating corpora for KWS was described, and a model for this task was evaluated. The model was designed to recognize only one keyword: "Fairy", pronounced by Japanese speakers. The authors created four types of testsets: positive data, random words, phonetically-balanced sentences, and "adversarial" words. All the classes except for the positive data were meant to test the model against the FPR. "Adversarial" words were created by inserting, deleting, or substituting a single phoneme or swapping two phonemes in the target keyword. All the sequences of phonemes that were impossible to pronounce were discarded.

A couple of testsets for speaker verification and keyword detection were based on a reduced number of wake-up words. In three parts of the RSR2015 [54] database for speaker verification, the speakers pronounced 30 fixed sentences from the TIMIT [55] database, 30 short audio control commands, and digit sequences. In [31], evaluation was done on GSC and two small in-house command testsets (10 and 5 commands, respectively). Only four keywords were tested in [37], and only two keywords in two Chinese testsets: [56] and [57].

2.2. End-of-speech Detection

The problem of EOS detection has already been approached with various methods. The simplest is based on a voice activity detection (VAD) model. The model is trained to classify frames as containing speech signals in this case. Once a frame (or a sequence of frames) not containing speech is detected, the pause event is activated. The threshold is set for the maximum pause duration, after which the system decides that the utterance is finished. For example, this type of approach was considered in [58]. While this method might work in the case of phrases that are spoken fluently (for example, in read speech), in the case of commands spoken spontaneously, humans tend to pause in the middle of a command for various amounts of time. Hence, choosing a threshold length that will not cause frequent cut-speech errors, i.e., premature closing of the microphone, is challenging.

EOS detection is often treated similarly to other classification problems for an audio stream realized on a frame-based level. In this case, a set of features is extracted and processed by a classifier for each frame. The classification can be based on prosodic features, as in [59] or [60], where an SVM model performed the detection using features such as periodicity, speaking rate, spectral constancy, duration/intensity, and pitch of prepausal speech. Unfortunately, these models were not evaluated against noisy conditions.

Other researchers have shown that prosodic features alone are insufficient to precisely describe the EOS: most studies use these features and different data types. This was the case in [61], where a decision tree model was fed with prosodic features accompanied by information from the language model. A similar solution was proposed in [62], where the detection model was trained based on broadcast news audio data. Prosodic features were processed by a DNN and mapped to boundary/non-boundary probability outputs. Those posterior probabilities were combined with lexical features and processed by a linear-chain conditional random field model (CRF) to make the final decision. CRF model for EOS detection was also proposed in [63]. In this case, the model processed prosodic and textual features. The proposed solution was evaluated on the task of lecture speech EOS detection. Such solutions required running the ASR decoder and an EOS detector, which is unacceptable in most online use cases.

In [64], log mel filterbanks were calculated for each frame and processed by a neural network composed of convolutional and LSTM layers. The authors compared their solution with a basic VAD-based approach. Both models were trained with voice search-specific data and force-aligned to generate the ground truth data. The results proved the superiority of the neural network model compared to the basic VAD-based approach – using their in-house corpus (15 k voice search-specific utterances), they showed that the neural network model yielded the median latency (EP50) approximately 100 ms lower than the VAD-based model while keeping the early EOS score at the same level. The same improvement was observed for the 90^{th} percentile latency (EP90).

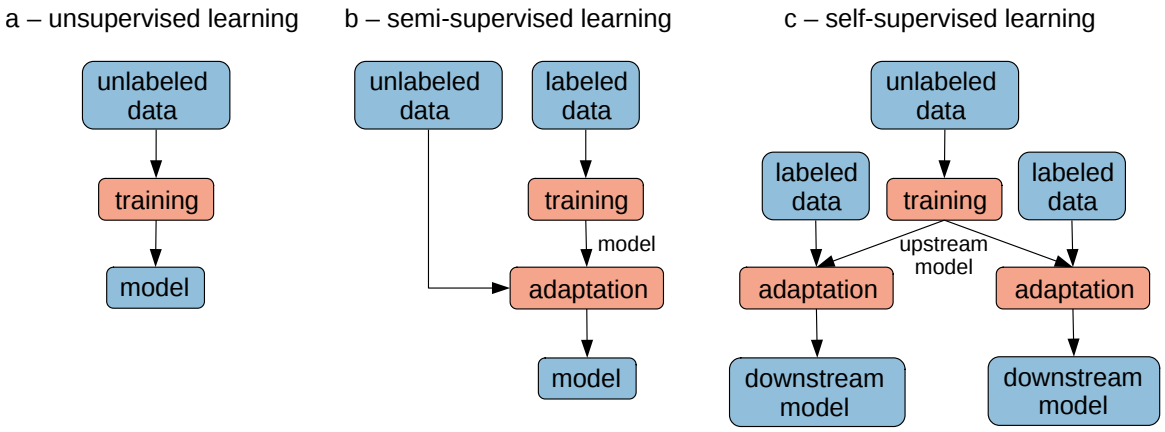

Figure 4. Overview of the machine learning methods utilizing unlabeled data.

2.3. Semi-supervised Learning for Speech Recognition

Machine learning is set on two cornerstones: models and data. The most popular paradigm in speech processing tasks is supervised learning. However, with this approach, the quality of the model depends heavily on the data used during the training. There are also other training methods utilizing unlabeled data (Figure 4):

- **Unsupervised learning** aims to discover unknown dependencies in the data through methods such as clustering, anomaly detection, and dimensionality reduction. Unsupervised learning has limited application in speech processing.
- **Semi-supervised learning** is based on labeled and unlabeled data. The labeled data is used to train a baseline model, which is further adapted with the unlabeled data. The unlabeled data is sometimes already used at the base model training stage.
- **Self-supervised learning** is also based on labeled and unlabeled data, but the training procedure is reversed. In the first step, a generic model is trained using unlabeled data. This model can be subsequently used in the adaptation for multiple downstream tasks. Such adaptation usually requires replacing the model's output layer and providing task-specific labeled data. Experiments show that the labeled datasets can be relatively small.

Note that there is an unfortunate collision in the names of the training methods. Throughout this thesis, I will abbreviate semi-supervised learning as SSL, while there will be no abbreviation for self-supervised learning.

Self-supervised learning has become very popular recently due to its ability to discover good data representation. Models trained with this approach can be used as encoders in numerous sequence-to-sequence tasks. Downstream adaptation consists of fitting the decoder to the task-specific labeled data. A thorough review of those methods can be found in [65]. Furthermore, multiple suggestions on the applicability of self-supervised learning to new areas can be found in [66]. There are numerous pre-trained upstream

models available in the public domain. Exemplary models for tasks related to speech processing are wav2vec [67], [68] and Whisper [69].

Large amounts of unlabeled data are necessary to train the upstream model in a self-supervised approach. However, I was interested in utilizing small datasets. This motivates the focus on semi-supervised learning, which uses unlabeled data to adapt a model instead of training it from scratch.

SSL has been used in solutions to multiple problems. Thorough surveys presenting such methods can be found in [70]–[73]. A detailed review of selected SSL algorithms can be found in [74]. This paper also compares the results of the described methods in image recognition tasks. SSL algorithms can be divided into three classes:

- **Consistency regularization** where the intuitive goal was to assure that realistic perturbations of the unlabeled data points would not change the model's output. Examples of this method were the Π-Model [75], Temporal ensembling [76], and Virtual adversarial training [77]. The last method is the most interesting since it proposed to modify each data sample in such a way that would most significantly affect the model's output.
- **Entropy-based methods** [78] applied the unlabeled data already at the training phase. The idea was to extend the loss function computed in a standard way over the labeled samples with values obtained for the unlabeled data. The additional term was designed to encourage the model to make "confident" (low-entropy) predictions for all examples.
- **Pseudo-labeling** [79] is very popular in practice due to its simplicity. The idea is to generate labels for the unlabeled data by the model itself and perform adaptation in a standard supervised manner.

SSL was also applied for the speech recognition task. In [80], an additional data selection step was proposed before the adaptation. The motivation was to exclude the samples, which would not bring new information to the model. The unlabeled data consisted of a different domain from the training data. Two scenarios were considered: when the in-domain labeled data was available for pre-training and without such data. SSL with data selection strategy introduced 17 % gain relative in the former scenario and 3.7 % in the latter.

Noisy Student Training was applied to the speech recognition task in [81], [82]. The idea was to mix small labeled and large pseudo-labeled datasets during training. Using 100 h subset of LibriSpeech and the remaining 860 h of this dataset as the labeled and unlabeled data respectively, the authors obtained WERs 4.2 %/8.6 % on the clean/other testsets. Experiments performed by extending the unlabeled dataset to LibriLight (60 k hours of speech data) lowered this metric to 1.4 %/2.6 % on the same testsets.

Extended experiments with SSL methods were performed in [83]. This work aimed to compare different contemporary speech recognition model architectures and loss

functions when adapted with pseudo-labels. Another set of experiments utilized different sizes of the unlabeled data in adaptation. The baseline model was trained with LibriSpeech, and the unlabeled data was extracted from LibriLight [84] (1 k, 3 k, 10 k, and 54 k hours of speech data). Already, the 1 k experiment introduced an improvement in WER. Yet, the best results were obtained with the largest unlabeled dataset (54 k), reducing WER from 2.54 %/6.67 % on dev-clean/-other to 2.11 %/4.59 %.

Iterative pseudo-labeling was introduced in [85]. The idea was based on the intuition that the AM's performance should increase during the adaptation; hence, the quality of the pseudo-labels should also improve. Thus, the pseudo-labels were generated iteratively. This method proved to have a better impact on the model when compared to the vanilla SSL (generating pseudo-labels once at the beginning of adaptation). The experiments were performed with subsets of LibriSpeech as the labeled dataset and LibriVox [86] as the unlabeled dataset. Once again, the results on WER improved with the amount of unlabeled data used in the adaptation.

SSL variant based on two models adapted simultaneously was proposed in [87]. One of the models was called "online" and was trained to predict pseudo-labels generated by the second model (called "offline"). The "offline" model's weights were calculated by the momentum-based moving average of the "online" model's weights. The models were trained using subsets of LibriSpeech and TEDLIUM3 [88] dataset. The best results were obtained using the largest unlabeled dataset.

The multilingual speech recognition model was adapted using pseud-labeling in [89]. The baseline model was trained using the MCV dataset (60 languages). In the following steps, the model was adapted for different languages. Experiments proved that using only pseudo-labels gave poor results for under-resourced languages. However, performing additional adaptation steps with labeled data improved the results significantly.

In [90], the effectiveness of semi-supervised techniques was analyzed, and it was suggested that the quality of the final result could be high even with the initial model being relatively poor. However, the experiments were performed using additional LM. The authors proved experimentally that the quality of LM has a much more significant impact on the SSL utility than the initial AM. With a carefully chosen recipe, they were able to reduce WER by 50 % relative, even when the initial system's WER was more than 80 %.

3. Contribution of this Thesis

This thesis investigates methods to improve the performance and accuracy of models applied to the audio stream in any of the steps of the speech transcription process in the on-device context. Such improvements can be obtained on various levels:

A) model architecture level,
B) model training or adaptation level,
C) inference level.

Levels B) and C) should be considered algorithmic improvements since they do not affect the models' architecture. In this thesis, levels A) and B) are considered.

Furthermore, the possibility to run the inference on-device was considered the critical factor throughout the research presented below. This assumption enforced strong constraints on the models' sizes and number of epochs in the adaptations.

In this work, the following theses are formulated and experimentally proved:

1. **The performance and accuracy of a keyword spotting model can be significantly improved by using a unigram language model and audio recordings generated by a text-to-speech system.**
   This thesis refers to the QbyT KWS problem and any AM computing frame-level posterior-gram. It should be noted that both improvements can be applied independently. They are applied on the model architecture level since they introduce additional layers after the AM. Both layers are constructed during the initialization phase. The unigram LM is a highly lightweight model, and TTS recordings usually generate only a handful of additional keywords. Hence, they allow more aggressive compression of the AM and deployment of the entire model to mobile devices. During the experiments, the AM compression was performed by reducing neural network layer sizes and applying post-training weights quantization. The critical factor in the unigram LM method is the model's weights initialization. A very simple yet powerful method of such initialization is proposed and evaluated. Recordings generated by a TTS model are used to improve performance on rare keywords not known during AM training.
2. **The accuracy of an end-of-speech detection model can be effectively improved by training the model with the proposed loss function.**
   This improvement is on the model training level since it uses a modified loss function. The basic procedure for EOS model training was to apply audio frame-level binary cross-entropy to compute model loss. However, adding weights to this function positively boosted such training. The critical factor in the proposed function is the choice of the weights. They are based on the frame distance from the actual end of the speech signal. This type of loss function is straightforward to compute and can

be applied to a wide range of model architectures. Especially those models that are small enough to fit on the mobile device can benefit from the proposed loss function.

3. **Semi-supervised learning methods can be effectively used to adapt acoustic models even with small datasets.**
   This thesis applies to the model adaptation level. More precisely, it refers to the adaptation of the models used for audio decoding. Experiments with SSL methods prove that such adaptations can be successfully performed with small datasets. Furthermore, it was shown that those datasets could contain much different data from those used to train the baseline model. The difference between training and adapting sets can be observed with features such as background acoustic conditions, microphone type, speaker articulation characteristics, and also, to some extent, the distribution of the words in both datasets. Since the datasets used in the experiments were tiny (the total length of the recordings counted in hours) and the adaptations were short (number of epochs smaller than 50), this method can be applied to adapt the acoustic models on mobile devices.

Experiments related to Thesis 1 have been described in the article titled "Open vocabulary keyword spotting with small-footprint ASR-based architecture and language models" [91], presented by the author at the FedCSIS 2023 conference in Warsaw, Poland. This paper was awarded The 2023 Professor Zdzisław Pawlak Award in the category of Industry Cooperation. Evaluations of models designed in those experiments were based on the testset described in the article titled "MOCKS 1.0: Multilingual open custom keyword spotting testset" [92], presented by the author at the Interspeech 2023 conference in Dublin, Ireland. Research towards Thesis 2 has been described in the article entitled: "Improved weighted loss function for training end-of-speech detection models" [93] presented at the MoMM 2020 conference. Finally, the study of methods corresponding to Thesis 3 has been published in the article entitled: "Semi-supervised learning with limited data for automatic speech recognition" [94] presented at the RTSI 2022 conference. The following sections (Sections 4–6) contain extended versions of the descriptions published in the publications mentioned above.

4. Custom Keyword Spotting

Various attempts to solve the problem of KWS have been proposed. Unfortunately, many of the solutions were evaluated on the proprietary testsets, making a comparison between models impossible. As previously mentioned in Section 2.1, only a few public testsets are available to evaluate KWS models. However, those testsets do not allow for an in-depth evaluation of the models since they contain a limited number of keywords and they lack challenging negative examples of keywords.

I will describe our attempt to fill in this gap by proposing a public testset built upon data from LibriSpeech and MCV. The testset was named MOCKS: *Multilingual Open Custom Keyword Spotting Testset* and was made freely available to the research community. Therefore, such a testset can become an effective tool when evaluating or benchmarking KWS algorithms.

This section will also present a novel solution to the KWS task for the non-streaming mode. It is based on the acoustic model (AM) architecture. 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) to increase the probability of correct hypotheses. We also explored the idea of using recordings generated by a TTS model in the final decision step 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.

4.1. Requirements for an Optimal Custom Keyword Testset

Analysis of the previous work on KWS shows that the testsets used to evaluate models suffered from several flaws, e.g., a small number of keywords, keywords being very short, only positive samples available, negative samples not challenging, or testsets designed solely for offline evaluation. To create a testset free from these drawbacks, we first defined a set of requirements that, in our opinion, should be followed when building an optimal KWS testset. These requirements are presented below, alongside their justification. Any parametric values were set heuristically during initial experiments.

1. **Keywords should be selected among phrases with the phonetic transcription length p such that $6 \leq p \leq 16$.** Production KWS systems usually have preset requirements for the length of a keyword. Shorter phrases might result in a high false positive rate (FPR) since they are usually similar to many other phrases or might be contained in longer phrases. On the other hand, long keywords are impractical for the user. Furthermore, KWS systems are usually deployed on devices with limited computing power, hence the requirement to restrain keyword length.

2. **The testset should contain positive and negative samples for each keyword.** Testing with positive data is insufficient, as KWS solutions should also minimize FPR. Furthermore, the negative samples should be varied and challenging.
3. **Similarity between phrases should be measured with normalized phonetic Levenshtein distance,** which proved to be successful in other studies, such as [36]. It allows us to decide which phrases are similar or different and, consequently, find which phrases are difficult to distinguish.
4. **Keywords should be selected among the words with at least two occurrences.** This requirement assures that in the QbyE KWS task, each keyword will have at least two test cases consisting of pairs of samples in the positive part.
5. **Positive samples for each keyword should contain phrases with the same phonetic transcriptions** to measure the true positive rate (TPR).
6. Negative samples for each keyword should contain:
   - **Recordings containing "similar phrases", i.e., the phonetic distance between the keyword and the tested phrase is in the interval** $(0.0, 0.5)$**.** These samples should be the most challenging for the model since they would be pronounced similarly to the given keyword.
   - **Recordings containing "different phrases", i.e., the phonetic distance between the keyword and the tested phrase is in the interval** $[0.5, \infty)$**.** The goal of this type of recording is to ensure a low FPR on the general types of speech.
7. **The testset should allow for evaluating performance in noisy conditions and online mode.** Production systems usually work in a challenging environment with different types of background noise. Additionally, keyword spotters most often work in streaming mode. Some solutions assume non-streaming mode, but they receive recordings processed by end-point detectors, which do not work perfectly, either.

4.2. Proposed MOCKS Testset

4.2.1. Source Audio Data

Producing and validating new audio data is usually a costly and time-consuming process. However, numerous datasets are extensive and varied enough to select subsets of data for tasks other than speech recognition. For this purpose, we used LibriSpeech and MCV corpora.

LibriSpeech has become a standard dataset for speech processing tasks, primarily due to its size (960 h) as well as the variety of speakers and vocabulary. However, the recordings in this dataset are very particular since they are extracted from English audiobooks read by professional speakers, and each audio sample contains a single sentence. The vocabulary is specific, and the average length of the recording is higher when compared to other datasets (7.42 s in *test-clean*, 6.54 s in *test-other* and 4.94 s–5.33 s in MCV, depending on

the language). However, its large vocabulary makes LibriSpeech a good candidate for a custom keyword spotting testset when one extracts only short parts of the recordings. In MOCKS, we used data extracted from *test-clean* and *test-other* splits. We will refer to the resulting testsets as *en_LS_clean* and *en_LS_other*, respectively.

MCV is another large dataset available in the public domain. It is based on crowd-sourcing and offers many annotated recordings in multiple languages. To prepare our MOCKS testset, we used Version 12.0 of the MCV dataset. We focused on five languages commonly used in Europe: English, German, Spanish, French, and Italian. We will refer to the resulting teststs as *en_MCV*, *de_MCV*, *es_MCV*, *fr_MCV* and *it_MCV*, respectively.

4.2.2. Creation of our Testset

We used an internally developed, rule-based grapheme-to-phoneme (G2P) algorithm to prepare phonetic transcriptions for each sample. Even though numerous phrases contained multiple variants of such transcriptions, we decided to use the most popular ones to reduce the number of compared phrases. In this case, their popularity was assessed by language experts.

The datasets designed for speech recognition tasks usually contain phrases with phonetic transcriptions that are much longer than the upper bound in our requirements. We used selected fragments of all phrases in the processed datasets to increase the number of potential keywords. We used word-level alignments generated by the Montreal Forced Aligner (MFA) and models available in the public domain [51] to extract audio data containing keywords. For each keyword, “similar phrases” and “different phrases” were selected so they would not include the keyword as a subphrase.

While creating “similar phrases” sets, we decided to use no more than ten phrases phonetically closest to the given keyword. Random selection was performed in case many phrases had the same phonetic distance. For each phrase, the “different phrases” set contains ten randomly selected recordings of the types described in the requirements above.

Our testset contains two versions of the audio samples: online and offline. For the

Table 2. Properties of MOCKS subsets based on LibriSpeech and MCV.

Property	**en_LS_clean**	**en_LS_other**	**de_MCV**	**en_MCV**	**es_MCV**	**fr_MCV**	**it_MCV**
# Keywords	6 883	6 918	6 581	6 534	7 694	5 540	8 062
# Total positive	97 924	115 334	159 890	105 126	195 246	101 764	208 626
# Total similar	195 360	200 350	182 305	178 747	271 258	166 338	266 288
# Total different	208 760	215 200	204 820	201 840	275 460	175 540	275 360
Offline min. len. [s]	0.23	0.35	0.33	0.34	0.24	0.32	0.34
Offline avg. len. [s]	0.80	0.79	0.83	0.89	0.80	0.86	0.90
Offline max. len. [s]	2.54	4.54	5.65	6.20	2.86	2.82	4.06
Online min. len. [s]	1.28	1.31	1.00	0.72	1.11	0.99	1.01
Online avg. len. [s]	2.89	2.87	2.82	2.87	2.75	2.80	3.08
Online max. len. [s]	5.72	6.88	8.48	9.20	7.54	6.60	7.59

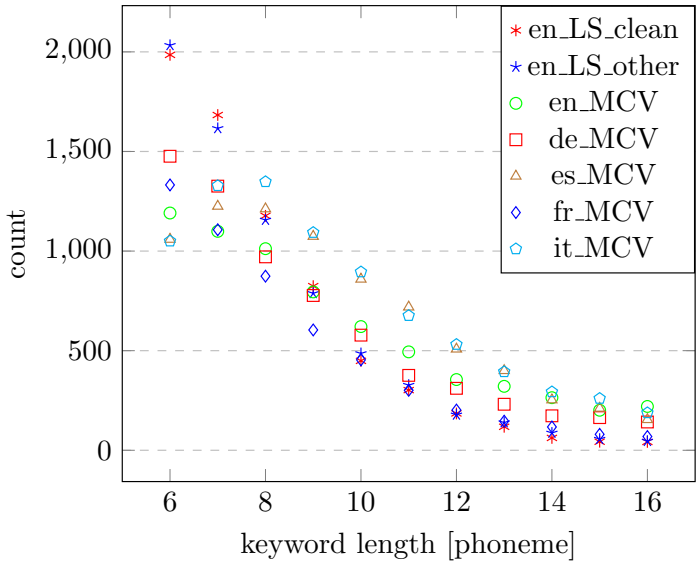

Figure 5. Distribution of keyword lengths in MOCKS subsets.

offline version, we used MFA-generated timestamps with additional 0.1 s at the beginning and end of the extracted audio sample to mitigate the cut-speech effect in the keywords. For the online version, we used MFA-generated timestamps with additional 1 s or so at the beginning and end of the extracted audio sample. This simulates the streaming mode since other words surround the keyword. The additional audio data might be smaller than 1 s if the keyword appeared at the beginning or end of the recording. If other words surrounded the keyword, the amount of additional audio data might be larger than 1 s since the cut was performed on the nearest aligned timestamp beyond 1 s. The online version of the testset contains timestamps of the keywords.

The testset preparation procedure's final step consisted of manually checking the transcriptions to exclude incorrect samples.

4.2.3. MOCKS Description and Analysis

Below, we describe in more detail the contents of MOCKS. In Table 2, several basic statistics on its subsets are presented, such as:

- **# Keywords** – number of keywords,
- **# Total positive/similar/different** – number of pairs in "positive/similar/different phrases" subsets,
- **Offline/Online min. len.** [s] – minimum keyword recording length in offline and online scenarios,
- **Offline/Online avg. len.** [s] – average keyword recording length in offline and online scenarios,
- **Offline/Online max. len.** [s] – maximum keyword recording length in offline and online scenarios.

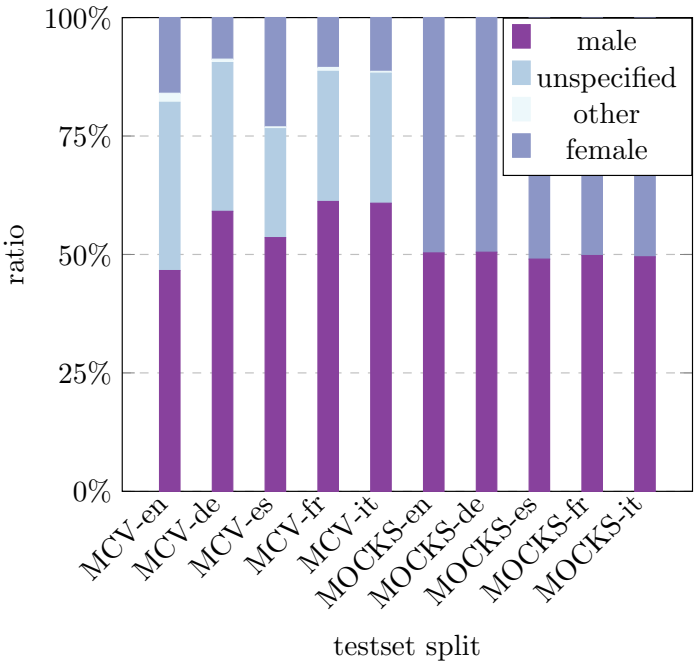

Figure 6. Gender distribution in MCV and MCV-originated MOCKS subsets.

Using the requirements described in Section 4.1, we obtained 5 000–8 000 keywords for each subset. Analysis of the keywords lengths is presented in Figure 5. Most of the keywords are short: in each subset, the keywords with six or seven phonemes constitute approximately half of all the keywords; hence, the average length of the recording in the *offline* scenario is under 1 s. The shortest recordings in the *online* scenario also have the length under 1 s, which is caused by the fact that the whole recording for the selected keyword was that short. Since *test-other* and *MCV* already contain data in challenging acoustic conditions, we decided not to mix the data with additional noise.

Both *en_LS_clean* and *en_LS_other* splits are balanced regarding speaker gender distribution. However, the original MCV datasets do not have this property: in most of the considered languages, nearly 60 % of the samples are marked as “male”, 8 %–23 % are marked as “female” and less than 2 % of the samples are marked as “other”; there is also a large number of samples with unspecified gender (see Figure 6). We randomly drew 2 500 “female” and 2 500 “male” samples for each language to generate keywords to remedy this issue.

The data is stored in a 16-bit, single-channel WAV format. 16 kHz sampling rate is used for *en_LS_clean* and *en_LS_other*, while 48 kHz for *en_MCV*, *de_MCV*, *es_MCV*, *fr_MCV* and *it_MCV*. This difference is a result of the source datasets’ sampling rates. Each testset split contains approximately 500 k test cases, which can be difficult to process, so we also add a subset of MOCKS to allow faster evaluations. Those subsets contain 20 k test cases in each scenario, and each testset split.

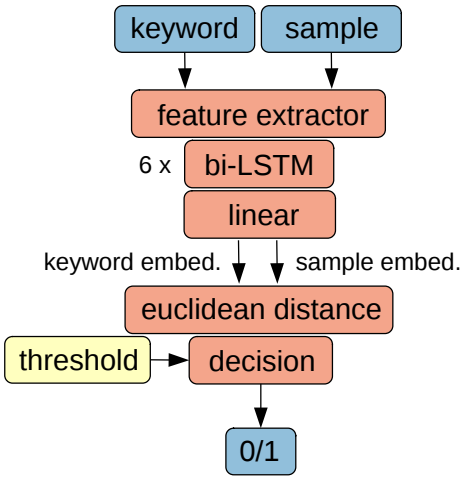

Figure 7. Architecture of the baseline QbyE KWS detection model.

4.3. Initial Experiments with MOCKS

To estimate the utility of the proposed testset, we evaluated a baseline model for the QbyE KWS offline task. Our model was based on the solution described in [95], and its architecture is presented in Figure 7. It consisted of an encoder with six bidirectional LSTM layers with 124 cells each and a linear layer generating 320-dimensional embeddings for audio data. The total number of trainable parameters was 2 M. We pre-trained the encoder model as a part of the Listen, Attend, Spell model [96] on the ASR task using all trainsets from LibriSpeech. Next, we appended the output layer to the pre-trained encoder and fine-tuned it for 20 epochs using the contrastive loss function. The dataset in the KWS fine-tuning step consisted of recordings generated by an in-house TTS solution from approximately 400 English keywords not included in MOCKS. The Euclidean distance between keyword and test sample embeddings was calculated and compared with a preset threshold during inference. We did not perform any hyperparameter fine-tuning.

To compare the results obtained on MOCKS and previously available testsets, we decided to use GSC, even though it was not prepared for the QbyE task. For each keyword recording in the GSC testset, we randomly selected 100 samples with the same phrase, 100 with a different phrase, and 100 samples from the “Silence” class. There were approximately 400 k test cases in each of those subsets.

Figure 8 shows the DET curves for GSC and MOCKS testsets. The DET curve for MOCKS was prepared after merging all the subsets of this testset. Appendix B contains additional figures with DET curves for different subsets of MOCKS (Figures 25 and 26). It should be noted that even though the model performed relatively well on GSC, the results were much worse on MOCKS. This clearly shows room for improvement and confirms how demanding MOCKS is. The estimated equal error rate (EER) value for all MOCKS subsets is 41.64 % with a confidence interval of ±0.15 % (at a 95 % confidence level).

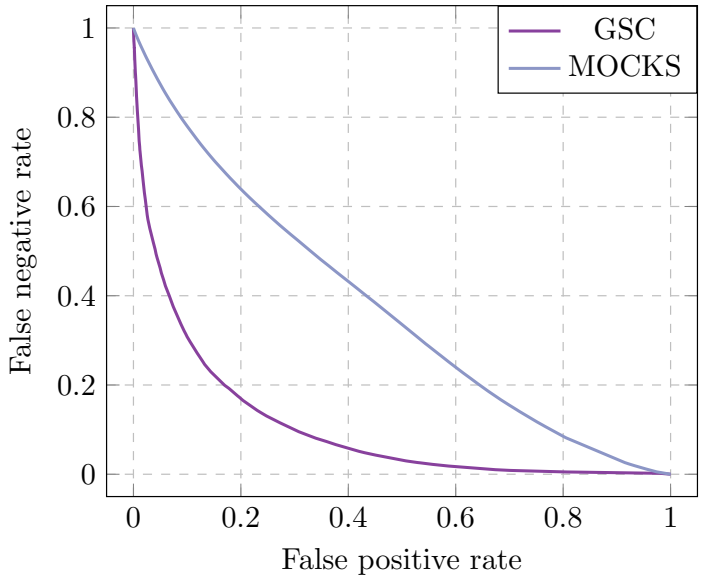

Figure 8. DET curve for MOCKS compared to GSC testset.

4.4. Query-by-text Keyword Spotting Model Architecture

After creating MOCKS and testing its utility on QbyE KWS, we moved our attention to another version of KWS: query-by-text, where the keyword is given by text. This task assumes 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. Figure 9 provides a general overview of our solution. We decided to adopt the AM architecture developed for the large vocabulary speech recognition task. AMs usually contain hundreds of millions of trainable parameters to gain high accuracy. 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. Our solution employed AM in a standard way to generate frame-level subwords. This was followed by a beam search to create the best path, which was converted to the final

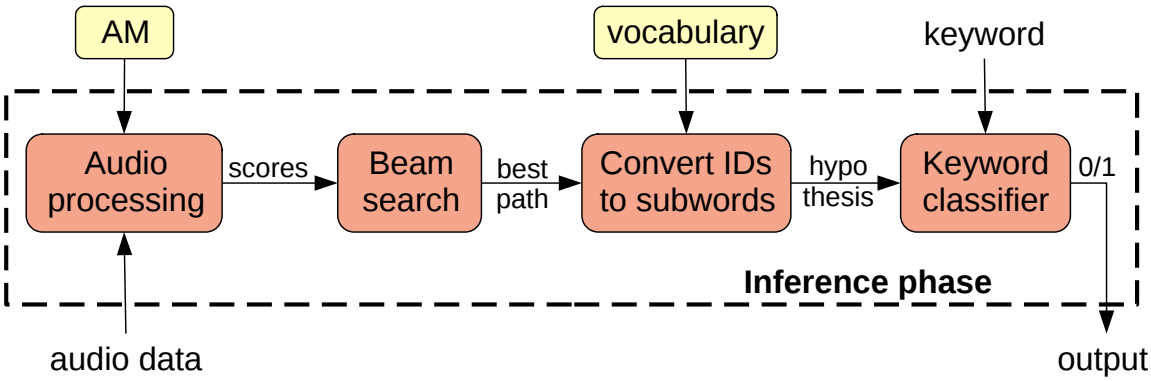

Figure 9. Overview of the baseline solution.

Table 3. AM architecture for QbyT KWS.

component	teacher	student
input	power-mels, 40-dim, 25 ms window, 10 ms step cepstral mean, 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 subwords	
parameters	50.1 M	3.1 M

hypothesis using model vocabulary. The last step consisted of keyword classification, which compared the hypothesis with the given keyword to decide whether a keyword was present in the recording.

4.4.1. Acoustic Model Architecture and Training

Our solution was based on AM implementing a monotonic chunkwise attention mechanism (MoChA) [97]. This type of model is based on the encoder-decoder architecture. In this approach, the encoder generates frame-level embeddings, which are processed by the decoder equipped with the attention mechanism.

We applied knowledge distillation to compress the model, as in [98]. There are two steps in this paradigm:

1. training large teacher model,
2. training 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 3. The input normalization parameters were computed over all training samples. The model's output

Table 4. AM training for QbyT KWS.

component	teacher	student
loss	CTC + cat. cross-entropy	distillation loss (weight 0.4) + cat. cross-entropy (weight 0.6)
training length	23 epochs	22 epochs
validation period	5 000 steps	5 000 steps
init. learning rate	1×10^{-4}	4×10^{-4}
learning rate decay	0.95	0.95
learning rate schedule	apply decay every 10 validation steps without change	
trainset	LibriSpeech + MCV + 4 h non-speech data	
trainset augmentation	noise + RIR + SpecAugment [100]	noise + RIR
quantization	✗	8-bit post-training

vocabulary was generated using the adaptation of byte pair encoding (BPE) [99] over all transcriptions from the trainset.

Table 4 describes the training procedure for teacher and student models. We used all the training splits from LibriSpeech and all the samples from MCV (version 7.0), which were not included in the testset. The data was mixed with background noise (with random signal-to-noise ratio in the range −2 dB–12 dB) and augmented with randomly selected room impulse response (RIR). RIR dataset contained simulations of distances from one to five meters and reverberation time between 0.2 s–0.9 s.

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*.

4.4.2. 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 unigram 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 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 decided to use a very simple 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 hyper-parameter. A general overview of the Static LM method is shown in Figure 10. The BPE algorithm was applied to the keyword with the same vocabulary used to convert the AM scores to obtain the hypothesis. Subwords included in the keyword were assigned a $boost$ weight, and all the remaining subwords were

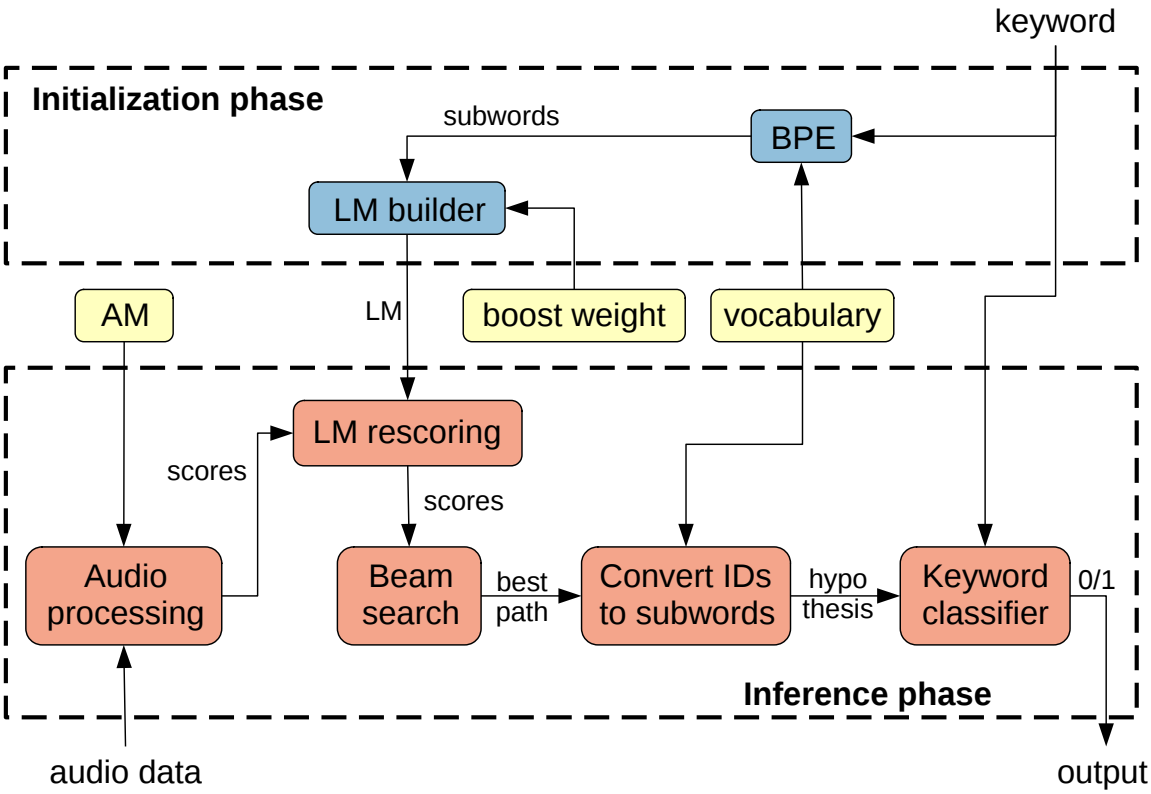

Figure 10. Overview of the solution with static LM.

assigned 1. Note that setting *boost* weight to 1 would not change AM scores, and setting *boost* weight to values smaller than one would decrease the probability of recognizing the keyword.

4.4.3. 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).

4.4.4. Multi-keyword Classifier

In the generic text-based KWS task, the keyword is provided by text. 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

Algorithm 1 Keyword classifier algorithm.

$keyword$ – custom keyword
hyp – hypothesis returned by AM
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 $dist(keyword, s) \leq t$ **then**
    return $true$
  end if
end for
return $false$

Algorithm 2 Multi-keyword classifier algorithm.

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

requires additional action from the user, we decided to pursue an automatic solution. An overview of this method is presented in Figure 11. We employed 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 system parameter. Each of those recordings was processed by the AM, followed by the beam search algorithm to generate a hypothesis. The original keyword was appended to the hypothesis list, and duplicates were removed. This list was treated as containing additional variants of the original keyword and was later used by the keyword classifier during inference.

Algorithm 2 presents the multi-keyword classifier pseudocode. It is the extended version of the keyword classifier described in Section 4.4.3. Once more, substrings of the hypothesis were selected for comparison, but this time, they were compared with each keyword from the list prepared during the initialization phase. As previously mentioned, normalized character level Levenshtein distance and a predefined threshold were used to make the final decision.

The main idea behind the multi-keyword classifier was to improve the recognition rate on keywords that were very distinct from the phrases presented to the AM during training. In such cases, the AM most likely would return an incorrect hypothesis. However,

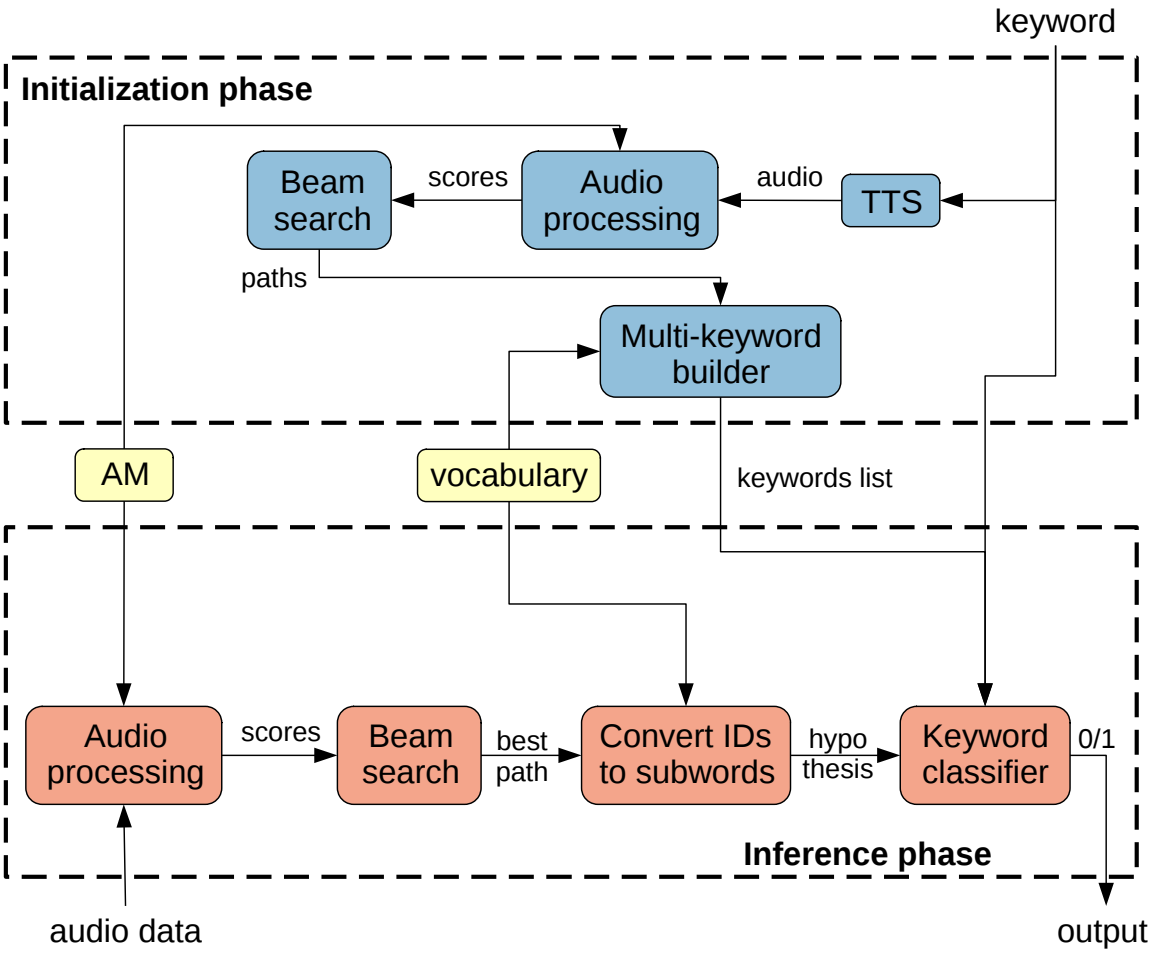

Figure 11. Overview of the multi-keyword solution.

provided those errors were similar across different samples of the same keyword, adding them to the classifier should increase the TPR. This procedure might also have negative results. Phrases similar to the keyword but different from it might be recognized with the same, wrong hypothesis. This would increase the FPR.

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

4.5. Experimental Results

4.5.1. Evaluation Procedure

Our solution was tested with the MOCKS 1.0 testset [92] and the 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 was composed of two audio files and a keyword. One of those files was treated as initialization (enrollment) phase data, and the other was treated as inference phase data. However, since our goal was to limit the user's interaction with the device, we skipped the initialization phase data for static LM. Furthermore, in the case of a multi-keyword classifier, we replaced 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*.

GSC is not suited for the open-vocabulary version of KWS since it contains few 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 [101]. 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 was designed for the generic case of open-vocabulary classification. To remedy this issue, evaluation for the negative test cases (labeled `_unknown_` or `_silence_`) was performed in the following way:

- In the initialization phase for each positive keyword, we prepared an LM and extended keywords list (if the multi-keyword classifier was enabled).
- After audio processing was done, for each positive keyword, we performed re-scoring, applied beam search, converted the result to the hypothesis, and finally computed the character level Levenshtein distance between the hypothesis and the given keyword.
- Finally, we found the minimal distance from the previous step. If this distance was less than or equal to the threshold, this sample was 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 supporting ten different voices. 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 [102]. The vocoder architecture was based on [103], and AM was similar to the Tacotron 2 [104] architecture as described in [105], with the use of the mutual information loss (MILoss) function [106]. Audio data included in The LJ speech dataset [107] and Hi-Fi Multi-Speaker English TTS Dataset [108] 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 full training data, followed by 450 k epochs of each voice-specific data.

For each keyword in MOCKS and GSC, we generated ten 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. 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.

4.5.2. Impact of the Boosting Weight

To test 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 considered in the beam search. We treat this case as the baseline solution.

Figure 12 shows the acceptance rate as a function of the threshold applied in the keyword classifier for *en_LS_clean*. Similar results for *en_LS_other* and *en_MCV* can be found in Appendix B (Figures 27 and 28).

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* were constant. This was because all keywords in MOCKS were relatively short (phonetic transcription length $p \leq 16$). As long as the $threshold$ was greater than 0.05, acceptance rates in both *pos* and *sim* increased. However, those values in the former subsets grew slower than in the latter. This means that increasing the threshold improved the TPR but increased the FPR even faster. Values of acceptance rate in *dif* started to increase only with $threshold > 0.4$ since this test set contained test cases that were very different from the given keyword. For such test cases, the AM returned very different hypotheses from the keyword, even if they did not match the proper transcription. Similar observations also apply to cases with $boost > 1$, except

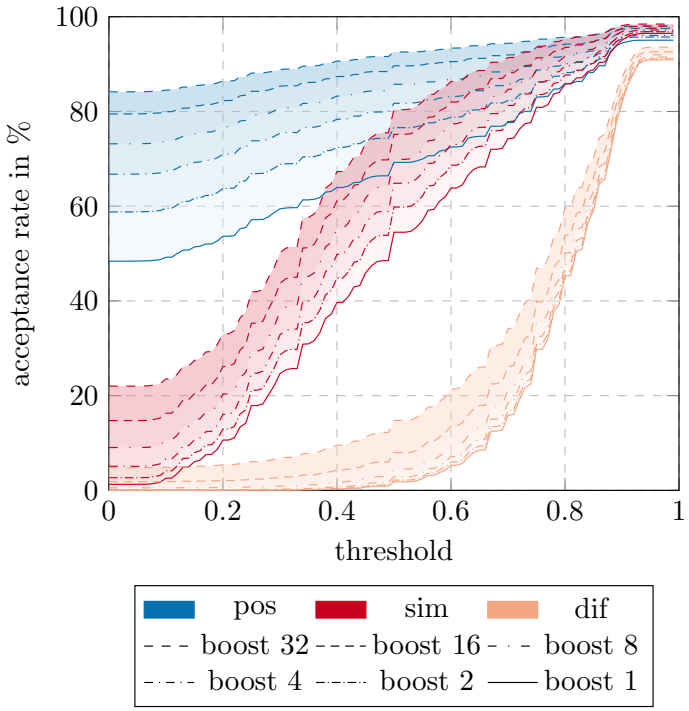

Figure 12. Boosting results with static LM, without multi-keyword classifier, for en_LS_clean testset.

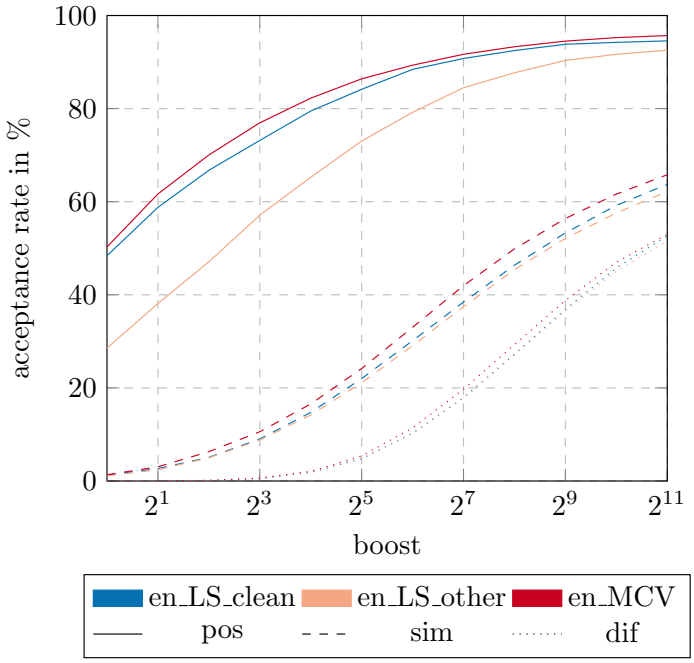

Figure 13. Boosting results with static LM, without multi-keyword classifier, threshold 0, for MOCKS testset.

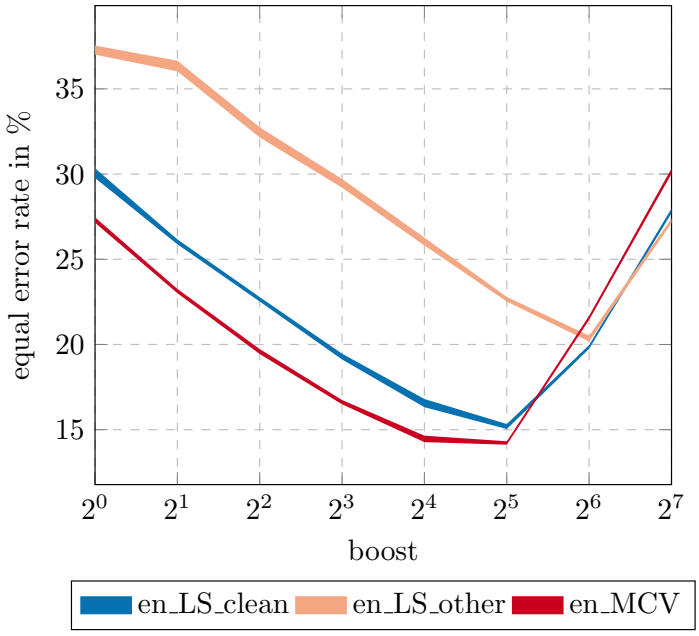

Figure 14. Boosting results with static LM, without multi-keyword classifier, for MOCKS testset.

for *dif*, for which the higher the $threshold$, the sooner this function started to grow. This analysis suggests that it is safe to use $threshold = 0$.

For clarity, Figure 12 shows evaluation results only for $boost \leq 32$. In Figure 13, we present evaluation results for all the testsets and $boost$ values up to 2 048 using $threshold = 0$. For $boost \leq 32$, for all the testsets, acceptance rates in *pos* were growing faster than in *sim*. Low acceptance rates characterized $boost \leq 32$ values in *dif* in all the testsets. However, the larger the $boost$ got, the faster acceptance rates in *sim* grew compared to *pos*. This was also accompanied by a rapid growth of those rates in *dif*. This observation suggests that there was a limit for $boost$, after which static LM brought more harm than benefit.

We used EER to estimate the optimal $boost$ value. For each testset, FPR was calculated after summing *sim* and *dif* subsets. Figure 14 shows EER for all the testsets and $boost \leq 128$ (for higher $boost$ values, EER was growing; hence it is omitted). The width of the lines in Figure 14 represent confidence intervals for EER estimation. It should be noted that the minimal EER values were located at $boost$ equal to 32 or 64, depending on the testset. The rapidly growing value of EER for large $boost$ was caused by the fact that in those cases, FPR was always more significant than the false negative rate (FNR). Since there was no point for which FPR and FNR were equal, the largest of those values was chosen as EER. Detailed values of EER for MOCKS can be found in Table 5.

Figure 15 shows the evaluation results for different $boost$ values on the GSC testset using $threshold = 0$. Applying static LM improved accuracy from 84.60 % for the baseline model to 95.97 % at $boost = 32$. For higher $boost$ values, accuracy dropped rapidly. This was because, with those large $boost$ values, the keyword subwords were favored

Table 5. 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 ± 0.28	37.27 ± 0.26	27.30 ± 0.16
16	16.56 ± 0.24	26.04 ± 0.23	14.46 ± 0.19
32	$\mathbf{15.18 \pm 0.14}$	22.65 ± 0.13	$\mathbf{14.22 \pm 0.10}$
64	19.86 ± 0.11	$\mathbf{20.34 \pm 0.17}$	21.57 ± 0.12

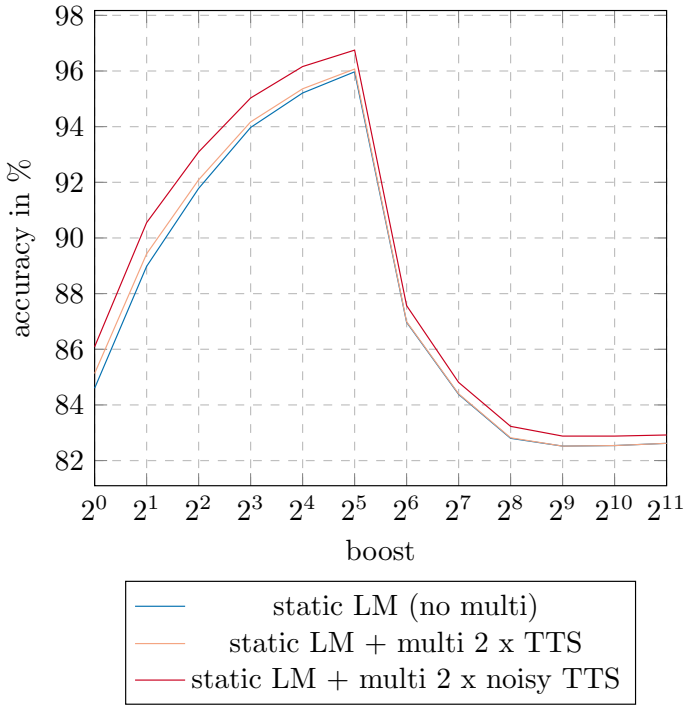

Figure 15. Boosting results with static LM, with and without multi-keyword classifier, for GSC testset.

during beam search. Therefore, the number of negative test cases that were recognized as keywords grew. Detailed accuracy values for GSC can be found in Table 6.

4.5.3. Impact of the Multi-keyword Classifier

The primary motivation for applying a multi-keyword classifier should be the increase in TPR, which ideally would not be accompanied by the rise in FPR. Our experiments show that this was not the case for MOCKS. Figure 16 shows the evaluation results using *en_LS_clean* for multi-keyword classifiers initialized with 2 and 10 clean and noisy TTS recordings. Those results are compared to a single-keyword classifier solution. Appendix B contains similar figures for *en_LS_other* and *en_MCV* (Figures 29 and 30). Furthermore, in Table 7, 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 4.5.2.

We observed that the improvement in acceptance rate on *pos* was more significant

Table 6. Accuracy in % for different methods tested 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**

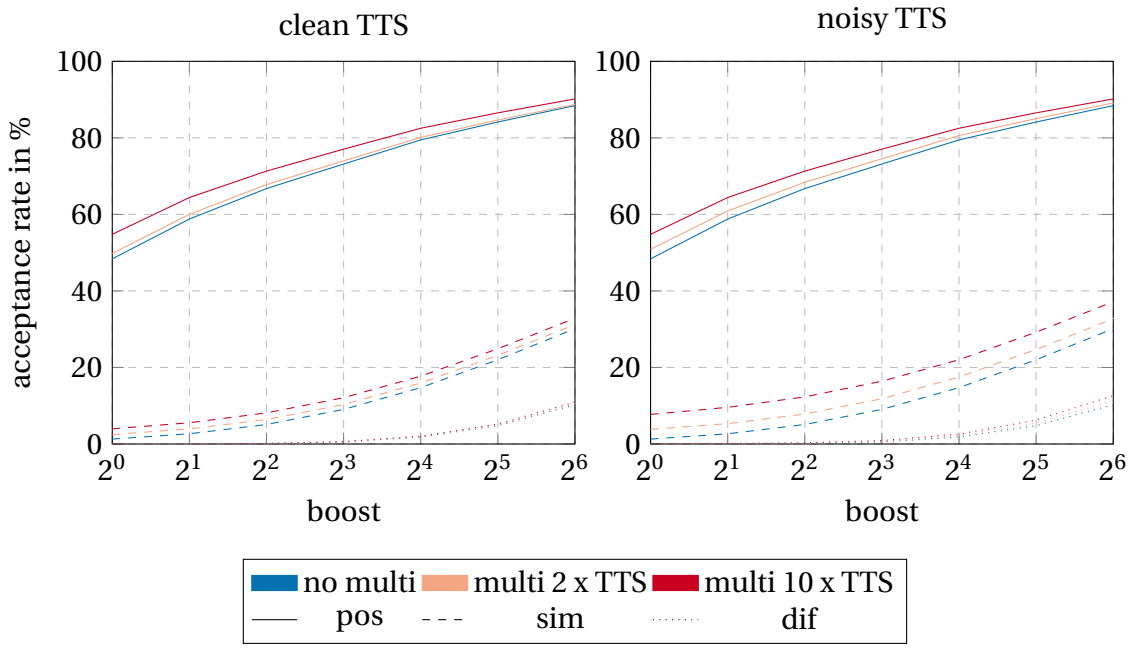

Figure 16. Boosting results with static LM, with and without multi-keyword classifier, for en_LS_clean testset.

than a similar increase on *sim* and *dif* only for small $boost$ values. This difference was minimal for the multi-keyword classifier initialized with two synthetic recordings. However, with ten such recordings, the improvement of the acceptance rate on *pos* was almost 2 % relatively higher than on *sim*. As soon as $boost = 8$, the 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 %–2.3 % relative depending on the $boost$). This improvement seems to be insignificant; nonetheless, it should be noted that it was gained in the range of 85 %–96 %. Even a minor increase in this metric is difficult to achieve at this level of accuracy.

Table 7. Acceptance rate in % for different methods on MOCKS.

method (boost value)	en_LS_clean			en_LS_other			en_MCV		
	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

4.6. Discussion

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 $boost$, the increase of FPR is much larger than the increase of TPR.

Using $boost = 32$ seems 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. Nevertheless, it should be noted that EER was minimal in *en_LS_other* with $boost = 64$; hence, this parameter has no universal value.

The positive impact of the multi-keyword classifier is especially visible with many TTS recordings generated 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 keyword variants is, the more likely the chance of false acceptance of phrases similar to the given keyword.

Evaluation of MOCKS and GSC with a multi-keyword classifier shows a significant difference in both testsets. With MOCKS at $boost = 32$, adding ten keyword variants increases FPR on *sim* more than TPR on *pos*. On the other hand, with GSC at $boost = 32$ and ten keyword variants, we still observed an improvement in accuracy. This can be explained by the fact that GSC contains short phrases, and the negative samples differ significantly from the keywords regarding the 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 % [50]. Still, it should be noted that our solution was designed for a far more complex task. GSC contains a very limited number of keywords, all very short and distinct. Finally, there are no challenging negative test cases in GSC. On the other hand, our solution was 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.

4.7. Conclusions

In this section, I described the Multilingual Open Custom Keyword Spotting Testset named MOCKS. This testset aims to provide a unified means of custom keyword spotting model evaluation and, in this way, to foster research on open vocabulary KWS solutions. This section also contains a list of requirements used while creating MOCKS. These requirements can be easily applied to develop new testsets in various languages based on other large vocabulary datasets. To create such testsets, two types of additional data are required: phonetic transcriptions and word-level alignment.

Once the testset for KWS was ready, it was possible to search for efficient models for this task. This was another topic that I tackled in this section. The result of this search was based on AM equipped with additional LM. The introduction of a simple unigram LM allowed for significant performance improvement. Furthermore, this type of LM can be used with different architectures of AMs.

5. End-of-speech Detection

Speech boundary detection (SBD) is usually performed at the very beginning of the audio processing pipeline. The precision of cutting the audio stream has a substantial impact on the remaining modules. As mentioned before in Section 1.3, SBD can be solved with two sub-tasks: Beginning-of-speech (BOS) and End-of-speech (EOS) detection. EOS is usually much more complex than BOS. This is especially true while processing the data collected from the users of virtual assistants. Often, people hesitate or make mistakes while speaking commands. This is visible as long pauses without speech, which are challenging for the EOS module. My goal in the topic of SBD was to improve the performance of a speech detection model in noisy environments, e.g., in the case of far-field speech. I decided to concentrate on detecting the end-of-speech (EOS) event, assuming that detecting the start of an utterance is performed by another module. Furthermore, I focused on improving the training procedure by extending with weights the function that is the most commonly used in this task.

5.1. Model Description

The model for EOS detection that we used in the following experiments was based on a binary classifier. As the input, we used feature vectors extracted from each signal frame, enriched by the past context. Our model was trained with 16-bit, 16 kHz audio data. Each recording was processed in a frame-by-frame manner, with no overlap, where the frames contained 256 samples, i.e., 16 ms of the signal (except for possibly the last frame of the file). We used 20-dimensional mel-frequency cepstral coefficients (MFCC) as the feature vector.

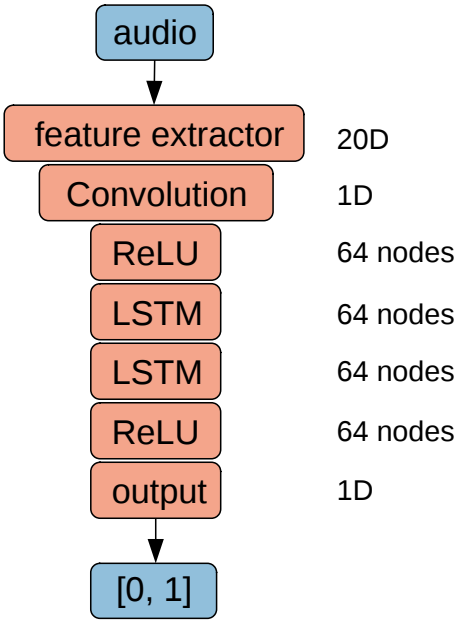

Figure 17. Architecture of the EOS detection model.

We used the architecture proposed in [64]. Figure 17 shows its schematic overview. The neural network contained a single one-dimensional convolutional layer, followed by a 64-node ReLU dense layer, two 64-node LSTM layers, and another 64-node ReLU dense layer. Such a combination of convolutional, LSTM, and feed-forward layers is called CLDNN [109]. The models that we trained were equipped with a 1-dimensional output layer. The output from this layer was interpreted as the probability of EOS. Each model was trained for 50 epochs. The decision of the classifier was made using the fixed threshold on the output layer:

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0.5 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

The model should return false (0) for all frames that appeared before the EOS. Note that such a condition implied that the model should also return 0 for the frames that did not contain speech but appeared at the stream's beginning. The same assumption was applied to all the silent frames between spoken words. The model should return true (1) for all the frames in the audio stream that appeared after the speech has ended. This means that the ground truth vector for every recording should match the regular expression: $0+1+$.

5.2. Proposed Loss Function

This section describes an improved loss function for training the EOS detector. Contrary to the binary cross-entropy loss function used by most machine learning-based detection algorithms, we proposed a weighted version that considers the distance from the EOS event.

Let us define this loss function H, calculated for the ground truth p and predicted probabilities q, as:

$$H(p,q) = -\frac{1}{N}\sum_{i=1}^{N} w_i \cdot (y_i \cdot log(\hat{y}_i)) + (1-y_i) \cdot log(1-\hat{y}_i))) \tag{2}$$

where y_i is the label of frame i in the ground truth data; $\hat{y}_i$ is the value returned by the model, interpreted as the predicted probability of frame i having label 1; w_i is the weight assigned to frame i and N is the number of frames for which the loss is computed (e.g., one batch).

The choice of weights was crucial for the method to work. We decided they should meet the following requirements:

A) set high weights for all frames marked as 0 (i.e., before EOS),
B) set low weights for frames marked as 1 (i.e., after EOS),
C) weights set for frames marked as one should be monotonically increasing.

The motivation for such requirements was as follows: detecting EOS too early tends to be highly problematic for most applications that use the output from the EOS detecting

module. Hence, during the training process, the model should be strongly penalized for returning values close to 1 before the actual EOS – this was guaranteed by requirement A). On the other hand, a slight delay in returning values close to 1 after EOS is acceptable; however, the penalty should grow with the distance from this event – this was guaranteed by requirements B) and C).

Therefore, we proposed the weights described by the following function:

$$h_w(i) = \begin{cases} \left\lfloor \frac{i - i_{EOS}}{k} \right\rfloor + 1 & \text{for } i_{EOS} \leq i < i_{EOS} + M \\ w & \text{otherwise} \end{cases} \tag{3}$$

where i is the frame index in the ground truth data, i_{EOS} is the index of the frame with EOS event, and w is the additional weight parameter that we will modify in our experiments. The w parameter controls the aggressiveness of the loss function. The higher this value is set, the more reluctant the model will be to output 1. M is the hyperparameter that controls the number of frames with lower weights. k is the hyperparameter that impacts the speed with which the weights grow after the EOS event. The higher the value of k, the slower the weights will grow after the EOS event. We assumed that $M \geq 0$ and $k > 0$.

By running a set of preliminary experiments, we heuristically set $M = 32$ and $k = 4$. Thus, the weights vector generated by the function defined in Equation 3 will look as follows:

$$(w \ldots w, \underbrace{1 \ldots 1}_{4}, \underbrace{2 \ldots 2}_{4}, \underbrace{3 \ldots 3}_{4}, \underbrace{4 \ldots 4}_{4}, \underbrace{5 \ldots 5}_{4}, \underbrace{6 \ldots 6}_{4}, \underbrace{7 \ldots 7}_{4}, \underbrace{8 \ldots 8}_{4}, w \ldots w) \tag{4}$$

Considering that the length of one frame was 16 ms, we decided that the period with a lower penalty during the training should not last longer than 512 ms, as extending this period would result in higher latency. Such an assumption was the motivation for setting M equal to 32 frames.

5.3. Experimental Setup

5.3.1. Data

We trained our model using the TIMIT corpus. Despite its age, it suited our task very well since it contains hand-verified phonetic transcriptions, adjusted with a signal sample precision. We used sample-level word transcriptions to prepare the ground truth for this dataset. The entire *TIMIT-train* was used in the training process. We split the *TIMIT-test* into two equal and disjoint parts and used one of them as the validating set during the training, while the other was used to evaluate the models. The evaluation set was not presented to the models during the training process.

Similarly to the majority of other works on EOS detection, we also used an in-house corpus collected in Samsung R&D Institute Poland in the evaluation process. This corpus will be called hereinafter INHOUSE. We used it to assess the generality of our models since this type of data had different acoustic characteristics from the training data and was not

Table 8. Characteristics of datasets used in experiments.

Dataset type	No. of utterances	Approximate No. of samples	Average samples No. per utterance	Average samples No. before EOS	Average samples No. after EOS
TIMIT train	9 240	2 933 k	317	179	138
TIMIT val	1 680	564 k	316	179	137
TIMIT eval	840	269 k	320	183	137
INHOUSE eval	9 753	3 261 k	334	171	163

presented to the model during the training. The in-house dataset contained free speech and was recorded in a far-field environment with natural acoustic echo. The distance between the audio source and the microphone was approximately 1.5 m. The ground truth for the INHOUSE corpus was generated based on the phonetic alignment generated by an ASR decoder. While the TIMIT contained recordings in American English, the INHOUSE dataset contained spoken commands in German, Spanish, French, and Italian, which should further test the model's generality.

All recordings in training, validating, and testsets were extended with 2 s of silence at the end. This was done to measure the model latency. Such a measurement would be impossible in the original recordings since they contained a marginal amount of frames after the actual EOS.

To test the performance of our model in a noisy environment, we mixed all the recordings with background noises. We used the LibriSpeech corpus for the noise set, which introduced a cross-talk type of noise. Both clean and mixed data were presented to the models during training, so the training and validating datasets were twice as large in frame numbers as the original ones. All the models were evaluated separately on clean and mixed data. We will refer to the testsets without background noise as “clean", while those mixed with LibriSpeech will be referred to as “LibriSpeech”.

Evaluation recordings were additionally mixed with two types of background noise, which was not presented to the model during training. For this purpose, we used Google AudioSet [110]. We used Speech and Music tags. We will refer to those modifications as “AudioSet Speech" and “AudioSet Music”, respectively. Table 8 briefly characterizes these datasets.

5.3.2. Metrics

Frame-level accuracy is a natural candidate for a metric used in binary classifier evaluations. Nevertheless, accuracy alone would not be informative enough as an assessment method in the case of EOS since only a limited number of frames just after the end-of-speech event is crucial. It could happen that the model would return correct answers for all the frames but one at the very beginning of the stream, as a result truncating

the stream at the beginning. This would mean the accuracy would be very high, but the model's response would be utterly useless as it would discard all the information from the stream. On the other hand, models returning only false values would score high accuracy while also being useless. Therefore, we decided to use other types of metrics (adopted from [64]) to evaluate our model:

- *Early EOS* – the proportion of the recordings in which the model mistakenly returned true before the actual EOS event, as a percentage.
- *Fine EOS* – the proportion of the recordings in which the model correctly returned true during or after the actual EOS event, as a percentage.
- *Fail EOS* – the proportion of the recordings in which the model mistakenly did not detect any EOS event, as a percentage.
- *EP50* – median latency over all utterances, in ms,
- *EP90* – 90^{th} percentile latency over all utterances, in ms.

EP50 and *EP90* were the main metrics used in [64] to evaluate models. Those are metrics based on the latency. We decided to use additionally *Early EOS*, *Fine EOS*, and *Fail EOS*, which are more similar to accuracy.

The *Early EOS* metric was defined rigorously: even if the model returned a positive value for a single frame before the actual EOS, it would be considered an error. Also, the *Fail EOS* metric was defined strictly: if the model showed too high latency, its response would be classified as a failure. This was motivated by considering the user experience: if the model had too high latency and detected EOS too late, users would have to wait too long for their command to be processed. Latency measured by *EP50* and *EP90* was defined as the difference in time between when the EOS occurred and when the model returned the first positive value. Note that this value can be negative for utterances counted as the *Early EOS*. Additionally, we set the latency to infinity in case of the *Fail EOS*. Hence, the *EP50* and *EP90* measure the typical and tail latency, respectively. We observed that the models did not detect any EOS at all for several test cases, which was reflected by higher *EP90* values. This motivated us to use the *Fine EOS* and *Fail EOS* metrics to show precisely the ratio of successful and failed test cases.

For confidence interval estimation, we used bootstrap resampling of the testsets. Each testset was resampled 200 times with replacement. Evaluations were performed with such resampled testsets and metrics described above were calculated. The trainset and model remained fixed. Such a method was applied due to the long model training time. To provide a 95 % confidence interval, we calculated the $[2.5, 97.5]$ percentile boundaries. The maximum size of the confidence interval was ± 0.34 %. We omit those values in Table 9 to improve the readability of the results.

5.4. Experimental Results

To verify the EOS detection using the proposed loss function, we compared it with the generic method, using binary cross-entropy as the loss function. Therefore, we performed two types of experiments:

1. with no penalty for early or late activation and generic binary cross-entropy loss, similar to the one used in [64],
2. using our training method with the proposed weighted loss based on the distance from the EOS event.

We also researched the impact of the input size and weights w used in the loss function (see Equation 3) on the EOS detection. The results are described in detail in the following subsections.

5.4.1. Experiments with Input Length

We started our experiments using various lengths of the temporal size of the input. We checked the following lengths: 8, 16, 24, 32, 40, and 48 frames of context, both for the generic and proposed weighted loss functions. In the case of the weighted loss function, we used 10 as the weight for 0 in the ground truth. These experiments are summarized in Figures 18 and 19.

The results showed that *Early EOS* yielded lower results for models trained with the proposed weighted loss than models trained with the generic loss, while for the *Fine EOS*, the opposite was true. Unsurprisingly, *Fine EOS* grew with the input length for both model types: the more history the model could see, the better it was at predicting the EOS events.

We observed that the clean TIMIT was by far the easiest testset. Models trained with concise input gained excellent results in terms of *Early EOS* (below 20 % for input length 8) and *Fine EOS* (above 80 % for input length 8). However, adding background noise (cross-talk) made TIMIT slightly more challenging. With short input lengths, models trained with the proposed loss function suffered a significant increase in *Fail EOS*, accompanied by a similar drop in *Fine EOS*. Somewhat surprising was the observation that for input length 8, the model trained with the proposed loss function obtained a lower value of *Fine EOS* than the model trained with a generic loss function. However, the longer the input context was, the better the results were in both *Fine EOS* and *Fail EOS*, with the best results for input length 40 (approximately 80 % and 15 % respectively). Unfortunately, none of the models trained with the proposed loss function could gain better results than the generic models in terms of *Fail EOS* on noisy data. With the best models, the difference in this metric was 5 %–10 % relative.

TIMIT is a specific dataset containing recordings of speakers reading given sentences in clean acoustic conditions. This means the speech is fluent, and the breaks between words in sentences are relatively short. Furthermore, the models in our experiments were trained on this data type. Hence, the good results even in the case of models trained with

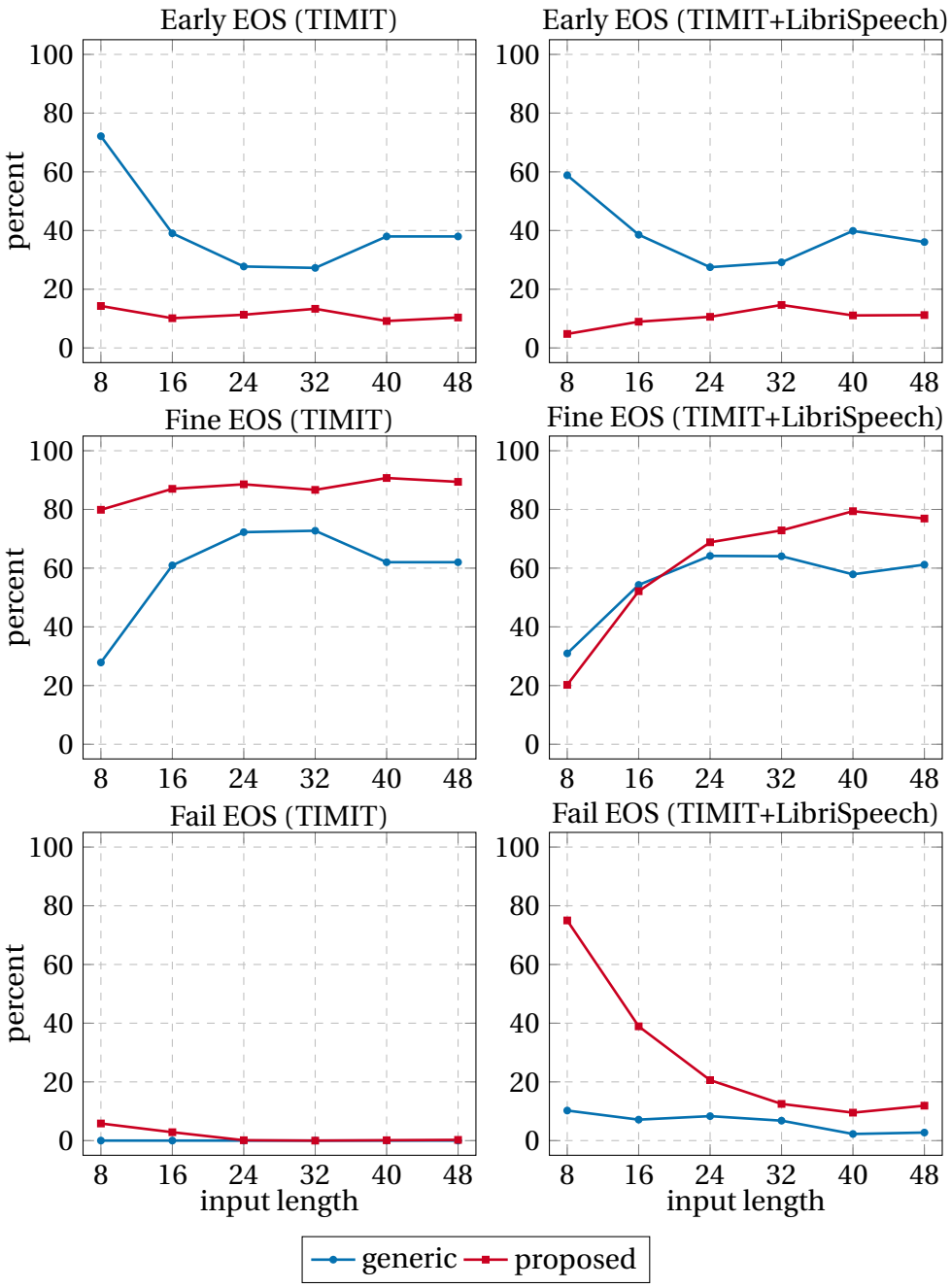

Figure 18. Impact of various input lengths on EOS detection for generic and proposed loss functions (for TIMIT and TIMIT+LibriSpeech testsets).

the generic loss function. Evaluations with the INHOUSE testsets (Figure 19) showed the actual usefulness of the proposed loss function. Models trained with the unweighted cross-entropy obtained high values in *Early EOS* and low values with *Fine EOS*. INHOUSE contains spontaneous speech, which is much closer to how people speak. Recordings have longer breaks between words, which trick EOS models into returning positive values too early, regardless of the input length. The introduction of the weighted loss function changed the results. With short input lengths, the models started failing to recognize

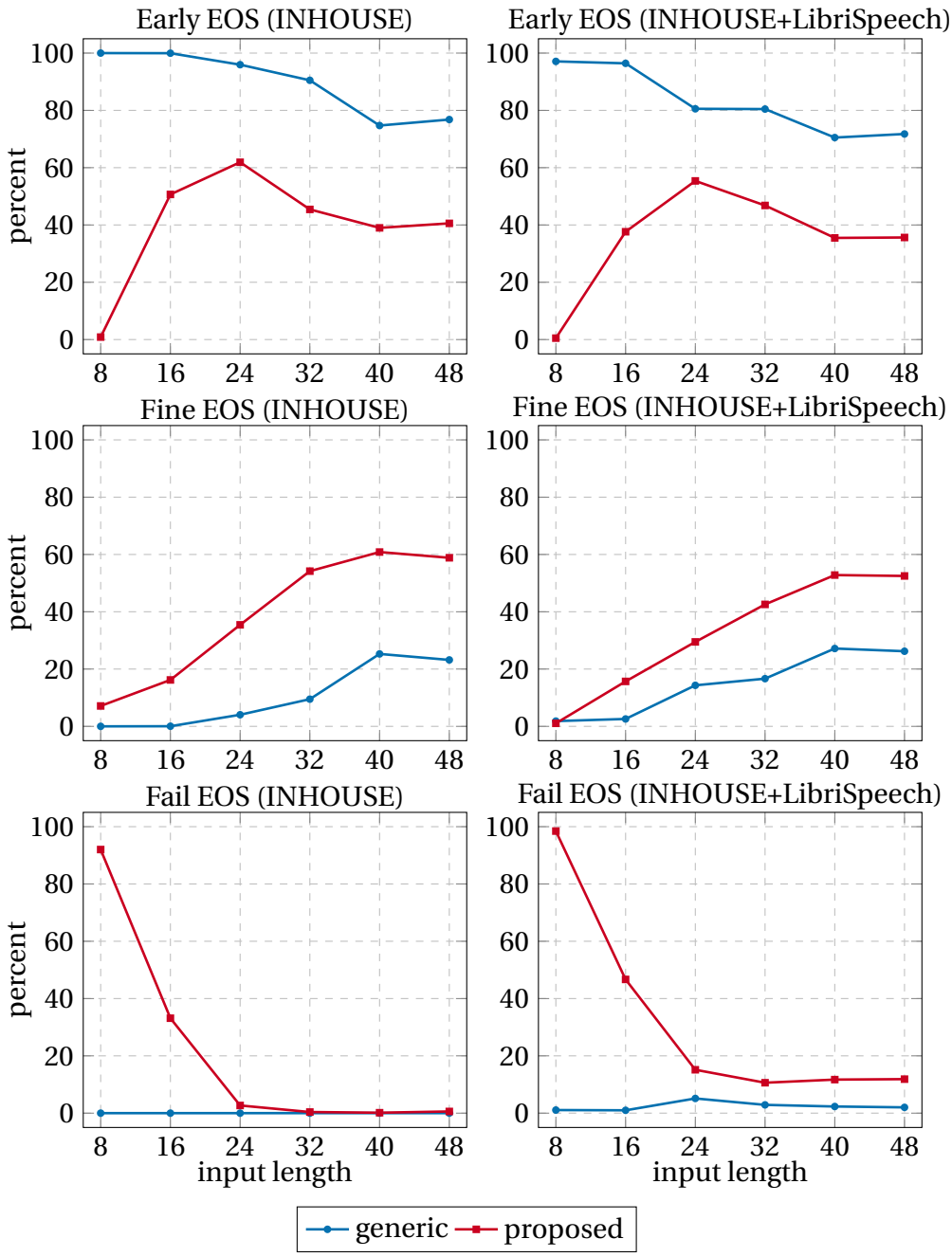

Figure 19. Impact of various input lengths on EOS detection for generic and proposed loss functions (for INHOUSE and INHOUSE+LibriSpeech testsets).

EOS at all (high *Fail EOS* values). However, providing the models with extended context fixed this issue, and *Fail EOS* dropped significantly. This change was accompanied by an increase in *Fine EOS* and *Early EOS*.

Adding cross-talk background noise to INHOUSE made this testset even more challenging. We observed that even with considerable input length, *Early EOS* and *Fine EOS* dropped, while *Fail EOS* increased. This means the models struggled to distinguish both

types of speech. This is an issue that could be solved with speech diarisation techniques. However, this is outside the scope of this thesis.

Based on these results, we decided to use 40 as the input length in the consecutive experiments since further increments of the input length did not improve the results. However, it might have a negative impact on a different aspect of the performance. Choosing the correct input is a trade-off between accuracy and speed since the longer the context, the more data that needs to be processed for each frame, and hence, the slower the system might become.

5.4.2. Experiments with Weights

With the most promising input size chosen, we moved on to training models with different weights for non-EOS frames in the weighted loss function (parameter w in equation 3). Figure 20 summarizes the evaluations of the models trained with various weights. We observed that adding a small weight did not improve the results when compared to the generic loss function. But as soon as the weight reached 10, the situation changed. *Early EOS* dropped, while *Fine EOS* grew rapidly for all the testsets. Increasing the weight values improved slightly *Fine EOS* in clean testsets (especially in INHOUSE data). However, with noisy testsets, *Fine EOS* started dropping in favor of *Fail EOS*. This tendency was visible for both clean and noisy data with weight 40, where *Fail EOS* on the INHOUSE testset reached almost 40 % and 60 % respectively. This means that the higher the weight, the more test cases were for which the model did not return any positive values. An intuitive explanation for this phenomenon could be that large weights set to non-EOS frames introduced a significant imbalance in the training data towards negative values; hence, the model was trained to output only zeros.

We concluded from these experiments that a weight equal to 10 assigned to the non-EOS frames in the weighted loss function was the best compromise between the *Early*, *Fine*, and *Fail EOS* metrics. We used this value to compare the performance of the models trained with the generic and weighted loss for various types of background noise. The results of these evaluations can be found in Table 9. When comparing the generic loss to its weighted counterpart for the clean testsets, we observed a remarkable increase in *Fine EOS*. At the same time, the *Fail EOS* score remained at a similar level, close to 0 %. In the case of testsets mixed with noise, we observed significant drops in the *Early EOS* (more than 50 % for all the testsets) and significant increases in the *Fine EOS*. Unfortunately, they were accompanied by the rise in the *Fail EOS* (by approximately 10 % relative) and in the *EP90*. Nevertheless, they seem to be minor compared to the improvements in the *Early EOS* (decrease from 58 % down to 25 % on average) and *Fine EOS* (increase from 40 % to almost 68 % on average) metrics. Such an increase in *Fail EOS* in noisy conditions can be explained by the model's tendency to classify background noise as the continuation of the utterance. This type of behavior was clearly visible in the cross-talk type of noise.

Notably, the *EP50* turned from negative in the case of the generic loss function into

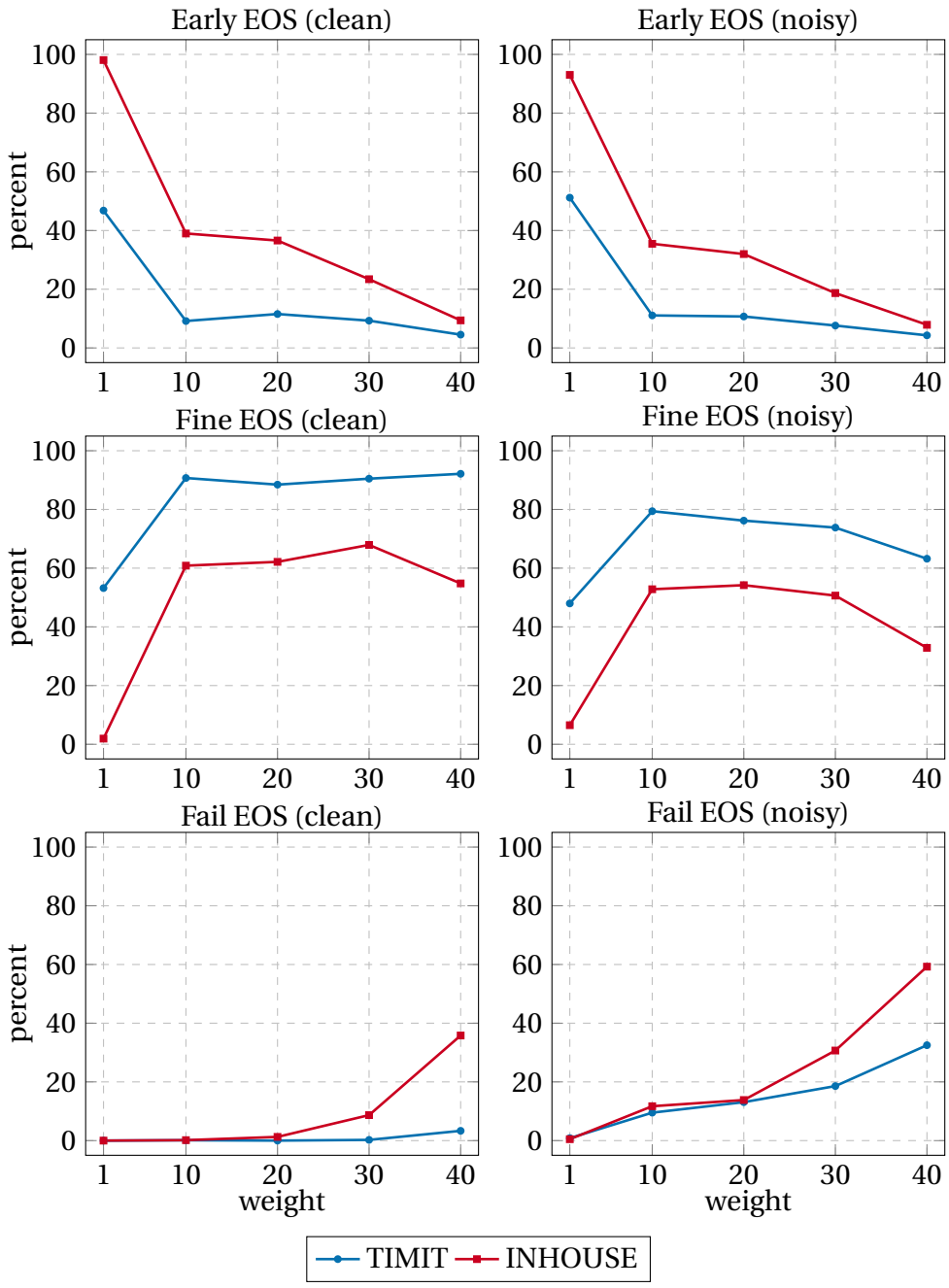

Figure 20. Impact of various weights on EOS detection for proposed loss function (for clean and noisy audio data).

positive for the proposed one, which means a drastic decrease in the cut-speech cases. This was also reflected by the drop in the *Early EOS* and the increase in the *Fine EOS*, as the model stopped returning positive values too early. We also observed that the *EP50* ranged between 100 ms and 200 ms, which means that the model was able to spot the EOS instantly once it happened. This observation was further supported by the *EP90* results, which ranged from 200 ms to 650 ms.

As mentioned before, the proposed *Early EOS* metric was very rigorous: even if the

Table 9. Evaluation of EOS detection for testsets mixed with various types of background noise (best results in bold).

Testset	Loss function	Early EOS [%]	Fine EOS [%]	Fail EOS [%]	EP50 [ms]	EP90 [ms]
TIMIT clean	generic	37.98	62.02	**0.00**	64	112
	proposed	**9.17**	**90.71**	0.12	128	240
TIMIT + LibriSpeech	generic	39.88	57.89	**2.26**	64	224
	proposed	**11.07**	**79.40**	9.52	176	480
TIMIT + AudioSet Speech	generic	46.79	50.36	**2.86**	64	192
	proposed	**13.57**	**76.35**	10.08	176	597
TIMIT + AudioSet Music	generic	53.10	44.64	**2.26**	-32	176
	proposed	**15.95**	**73.93**	10.12	160	625
INHOUSE clean	generic	74.71	25.28	**0.01**	-384	112
	proposed	**38.99**	**60.85**	0.15	96	240
INHOUSE + LibriSpeech	generic	70.51	27.17	**2.32**	-368	176
	proposed	**35.48**	**52.81**	11.71	128	640
INHOUSE + AudioSet Speech	generic	71.17	27.05	**1.78**	-368	160
	proposed	**38.11**	**52.49**	9.40	112	464
INHOUSE + AudioSet Music	generic	73.61	24.85	**1.54**	-432	144
	proposed	**38.88**	**53.52**	7.60	112	384
average TIMIT	generic	44.43	53.72	**1.85**	40	176
	proposed	**12.44**	**80.10**	7.46	160	486
average INHOUSE	generic	72.5	26.09	**1.41**	-388	148
	proposed	**37.86**	**54.92**	7.22	112	432
average TIMIT + INHOUSE	generic	58.47	39.90	**1.63**	-174	162
	proposed	**25.15**	**67.51**	7.34	136	459

model returned a positive value for a single frame before EOS, the test case was counted as failed and included in this metric. During our experiments, we observed that models occasionally produced very short sequences of positive values before the actual EOS event for some samples. Such samples would be counted as *Early EOS*. This could be fixed in the post-processing step by setting a threshold on the minimal length of such a series. All the series shorter than this threshold would be treated as negative values. However, it should be noted that such an approach would introduce additional latency.

5.5. Conclusions

In this section, I proposed an improved method of training models for detecting EOS events. It is based on the established model architecture but adds a new ingredient of weights to the loss function used during training, significantly improving EOS detection.

The system was trained on a clean read speech from the TIMIT dataset. The excellent performance of the proposed method was confirmed on more challenging far-field recordings of spontaneous speech from an in-house corpus and in the presence of noise from the AudioSet data.

6. Semi-supervised Learning for Speech Recognition

ASR models and machine learning tasks depend heavily on the amount and quality of data used during training. Even though large amounts of audio data can be gathered quickly, the transcription process is slow and usually expensive. To overcome this problem, semi-supervised learning (SSL) methods have been proposed. The goal of those methods is to use unlabeled data to improve the performance of a baseline model.

Numerous experiments prove that SSL is effective. However, there are known limitations to these methods. For example, it was shown that SSL is most effective in cases of large unlabeled datasets, exhibiting the same distribution as the labeled data used to train the baseline model [74]. A similar approach was used in the domain of ASR, where the size of unlabeled datasets was counted in hundreds or even thousands of hours [82], [87]. This constraint limits the use of SSL to server-side models. The client-server architecture has been the most commonly used approach in commercial applications based on ASR. This was the obvious choice since the computing power of end-user devices, such as smartphones, was too low to perform inference with large vocabulary continuous speech recognition models. However, on-device inference has become possible with the constant minimization of microchips and recent advancements in neural network architectures. Currently, the question about on-device ASR is not whether it is possible but rather how strong models can be deployed to end-user devices and even what modifications can be applied to those models. Hence, in this section, I would like to do the following:

- Verify whether it is possible to employ SSL with datasets limited in size, targeting, e.g., on-device scenario.
- Try to use SSL with data much different from the data used to train the baseline model.
- Propose simple improvements to the basic SSL method, which can be used in small datasets.

6.1. Proposed Method

The basic SSL method uses pseudo-labels generated by the adapted model itself [83], [111]. This method can be represented by Algorithm 3 with variables *update pseudo* (line 11) and *adapt baseline* (line 14) set to false. In this case, the set of labels is invariant during the entire training. This method will be referred to as *keep pseudo* in this thesis. It should be noted that in the generic version of the pseudo-labels method, a threshold is used to select only those samples for which the model generates a hypothesis with significant scores. This means that the confidence score of those labels is high enough. In the case of our experiments, the baseline (labeled) training data was different from the unlabeled data in terms of acoustics and vocabulary. Hence, the confidence score of almost all unlabeled samples was very low. Furthermore, unlabeled datasets were tiny by

Algorithm 3 Semi-supervised adaptation with modifications.

Input:

$Baseline$ – baseline model
U – unlabeled audio dataset
$G_p = \{(audio_i, ref_i)\}$ - labeled dataset for pretraining
$G_a = \{(audio_j, ref_j)\}$ - labeled dataset for adaptation
m – number of pretraining epochs
n – number of adaptation epochs

1: $M \leftarrow Baseline$
2: **if** $G_p \neq \emptyset$ **then**
3: **for** $i \in [0..m]$ **do**
4: $M \leftarrow adapt(M, G_p)$ ▷ pretraining
5: **end for**
6: $Baseline \leftarrow M$
7: **end if**
8: $L \leftarrow M(U)$ ▷ Generate pseudo labels
9: **for** $i \in [0..n]$ **do**
10: $M \leftarrow adapt(M, L \cup G_a)$
11: **if** *update pseudo* **then**
12: $L \leftarrow M(U)$ ▷ Generate pseudo labels
13: **end if**
14: **if** *adapt baseline* **then**
15: $M \leftarrow Baseline$
16: **end if**
17: **end for**

design. Applying a threshold on the hypothesis scores in extreme cases would empty the chosen sample sets.

A simple improvement to the *keep pseudo* method consists of iterative updating pseudo-labeling [85]. This idea leverages the fact that in each epoch, the model's output is becoming closer to the distribution represented by the data used for adaptation. Hence, the labels generated by the model after each training epoch should be closer to the ground truth. We used a similar approach to develop a new set of labels after each training epoch in Algorithm 3 with variable *update pseudo* set to true in line 11. We will call this method *update pseudo*.

6.1.1. Improvement 1: *update pseudo, adapt baseline*

In this work, we proposed additional improvements to the methods described above. First, we introduced the idea of training the baseline model (which is not overfitted yet) with the pseudo-labels generated by the model from the previous epoch. This concept is implemented with the *adapt baseline* (line 14) set to true and will be referred to as *update pseudo, adapt baseline*. We justify this approach by the fact that however good the labels generated by the model are, they can still contain errors. Furthermore, performing training using small datasets can quickly lead to model overfitting.

6.1.2. Improvement 2: *update pseudo, adapt baseline, ground*

Even with the help of *update pseudo* and *adapt baseline* modifications to the basic SSL method, the models sooner or later became overfitted to the pseudo-labels and degraded. To overcome this issue, we decided to use additional data which would be equipped with the ground truth. This idea is implemented in Algorithm 3, where G_a is the dataset containing ground truth. This dataset is added to the pseudo-labels set during the adaptation in line 10. Note that the additional data in this case was taken from a different dataset than the unlabeled data. In our experiments, we used a subset of the Mozilla dataset. We performed experiments with two flavors of this method: *update pseudo, ground* and *update pseudo, adapt baseline, ground*, where the former gradually adapts the same model throughout all the epochs. At the same time, the latter uses a baseline model for adaptation in each epoch.

6.1.3. Improvement 3: *update pseudo, adapt baseline, pre-train*

The last idea we wanted to explore was to loosen the constraint of using only the unlabeled data from the target distribution. In this case, the idea was to use a small dataset with ground truth from the target distribution to adapt the baseline model. This step was followed by adapting the model with pseudo-labels (possibly mixed with another labeled dataset). This approach is shown in Algorithm 3 with variable G_p set to a non-empty set of $(audio, reference)$ pairs.

6.2. Experimental Setup

6.2.1. Model Architecture and Data

We trained the baseline model using Monotonic Chunkwise Attention (MoChA) architecture [112]. This is one of the contemporary architectures based on the encoder-decoder approach. We trained it using the sum of CTC and Cross-Entropy losses. The encoder comprised a stack of LSTM layers, while the decoder contained two attention layers (monotonic and chunkwise) followed by an LSTM layer. Such type of attention in the decoder allows for online inference, which is crucial in production use cases. Furthermore, MoChA models can be strongly compressed without performance loss using methods such as the low-rank matrix approximation method by employing DeepTwist [113]. Online inference and strong compression make MoChA architecture a good candidate for on-device use cases.

To train the baseline model, we used 960 h of speech from LibriSpeech (*train-clean-100*, *train-clean-360* and *train-other-500* subsets) as the labeled dataset, obtaining the model which achieved the WER of 7.24 % and 19.23 % on *test-clean* and *test-other* LibriSpeech subsets, respectively.

It should also be emphasized that the characteristics of the training data in LibriSpeech are very specific since it contains sentences extracted from audiobooks. The average

Table 10. Statistics of data used in our experiments.

dataset	No. of rec.	unique sentences	unique tokens	total length [h]	avg. length [s]
LibriSpeech	281 241	281 071	89 114	960	12.3
TIMIT	4 620	1 735	4 919	4	3.1
Mozilla	50 000	37 890	39 148	70	5.0
en-SE	6 992	2 113	6 728	12	6.2
INHOUSE	7 500	6 037	2 303	7	3.2

recording length in LibriSpeech is much greater than the user commands in production systems, and the vocabulary distribution is entirely different. Obviously, production systems vocabularies depend heavily on the domain they support. Hence, it is challenging to prepare a generic model for different products. Therefore, to emulate various types of speech data, we decided to add the following datasets to our experiments: TIMIT, MCV, Southern English (en-SE) [114], and the INHOUSE dataset prepared by the Samsung R&D Institute Poland.

In the case of TIMIT, we used only *TIMIT-train*. Out of the MCV dataset, containing over 2000 h (1 271 213 recordings) of en-US audio, only 50 000 randomly selected recordings were used. We will refer to this subset as Mozilla throughout this section. The INHOUSE dataset contained a random selection of the most common commands, which an intelligent voice assistant processes. Those commands represent a specific vocabulary distribution and acoustic characteristic (far-field speech and heavy background noise).

Basic statistics of the abovementioned data can be found in Table 10. For reference, we have provided the statistics for LibriSpeech (all the *train* splits).

For each unlabeled dataset listed in Table 10, we used separate testsets. In the case of TIMIT, we used the *TIMIT-test*, while for Mozilla, en-SE, and INHOUSE, we randomly selected 1 000 recordings and excluded them from the adaptation dataset. Finally, in the case of en-SE and INHOUSE, we also randomly selected 500 recordings for the experiments with pre-training. Those recordings were treated as being manually transcribed.

Using this setup and data, we performed the SSL training using the basic methods and the methods proposed in Section 6.1.

6.2.2. Metrics

To evaluate our experiments, we used the token-level Levenshtein distance between the ground truth and the model's hypothesis averaged over all the samples (recordings) in the testset, known as WER in the ASR research. However, we also split this metric into three separate components:

- deletions (*Del*) – number of omitted tokens divided by the total number of tokens in the testset,

- insertions (*Ins*) – number of unnecessarily added tokens divided by the total number of tokens in the testset,
- substitutions (*Sub*) – number of misrecognized tokens divided by the total number of tokens in the testset,

so that $WER = Del + Ins + Sub$. By "tokens," we mean the longest continuous sequence of non-blank characters. Splitting the metric facilitated a better insight into the model's performance and allowed for optimizing its training process.

For confidence interval estimation, we used bootstrap resampling of the testsets. Since the adaptation was the most compute-intensive part of the experiment, we decided to perform it only once for each method. Thus, the unlabeled datasets and models remained fixed during the evaluation. Each testset was resampled 200 times with replacement (audio and transcriptions), and an evaluation was performed with such data. The average WER was calculated. To provide a 95 % confidence interval, we calculated the $[2.5, 97.5]$ percentile boundaries over all resampled evaluation results.

6.3. Experimental Results

Figure 21 presents evaluation results for different SSL adaptation methods. Models obtained in each epoch were evaluated with the testsets corresponding to the adaptation data.

6.3.1. Results for Fixed Pseudo-labels

The first method that we tested was based on fixed pseudo-labels generated by the baseline model (*keep pseudo*). We observed that in the case of TIMIT, the model's performance improved in the initial epochs but soon degraded and reached WER 7 % relatively higher than the baseline model. In the case of Mozilla, surprisingly, this method gave the best results, especially since the model had a very stable performance in the later epochs. Adaptations using the en-SE and INHOUSE datasets showed minor improvements in the initial epochs, but just as in the case of TIMIT, the models degraded in the subsequent epochs.

6.3.2. Results for Updated Pseudo-labels

The introduction of updating pseudo-labels after each epoch (*update pseudo*) had already improved the results for all the datasets but Mozilla. However, this was at the expense of model degradation after just a couple of epochs for all the datasets except en-SE, which remained stable for as many as 50 epochs. Such degradation was most visible with Mozilla, where the model degraded heavily already during the third epoch and reached WER twice as large as the baseline model.

Both *keep pseudo* and *update pseudo* methods rely on the gradual adaptation of the same model. Our experiments showed that resetting the model to the baseline after each pseudo-label update might give good results in some cases. This method (*update pseudo,*

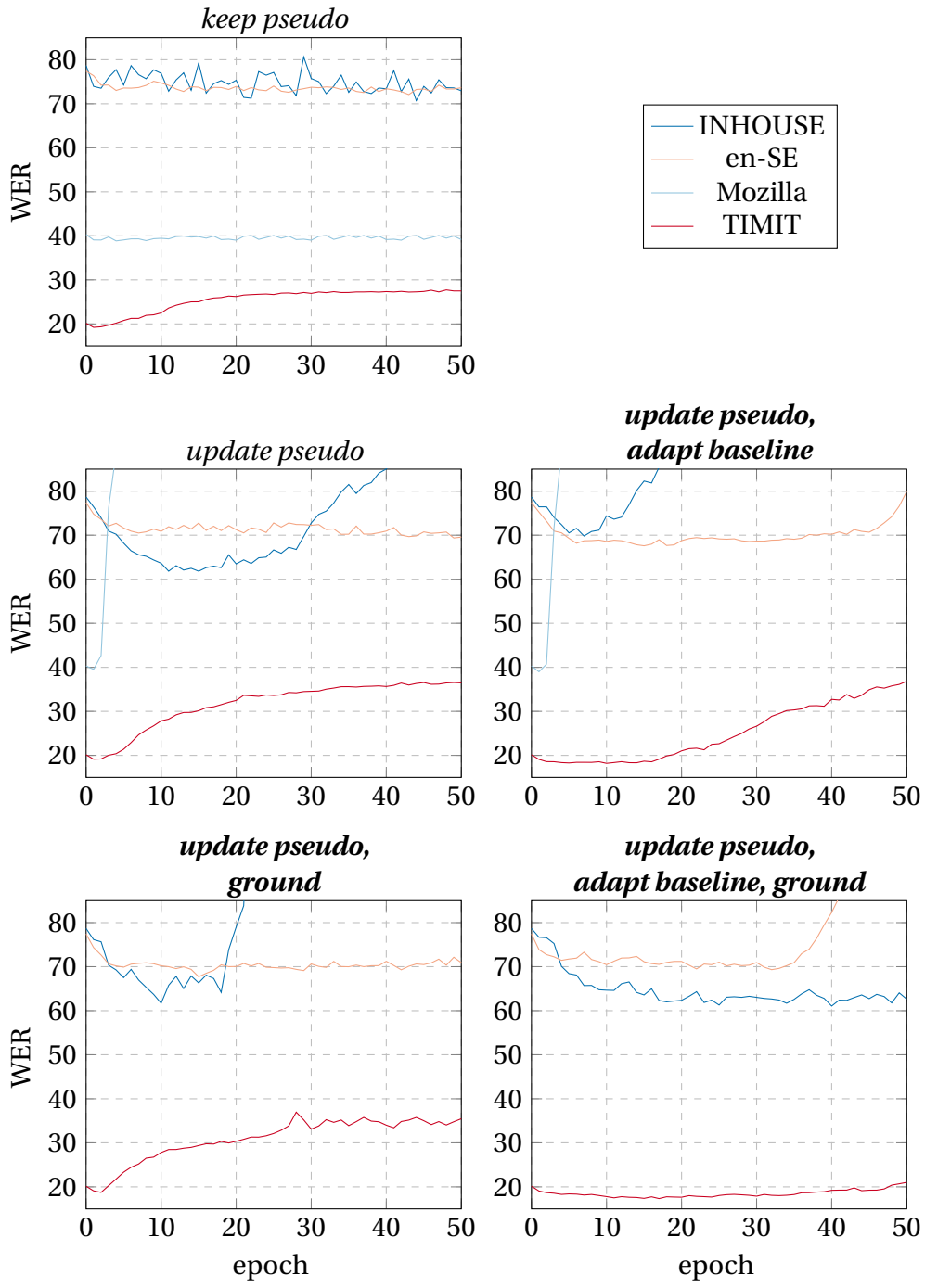

Figure 21. Evaluation results of different SSL methods (our improvements labeled in bold).

adapt baseline) used with TIMIT and en-SE produced good models. The former was stable for more epochs than *update pseudo*, while the latter gave the best results in all the experiments performed on this dataset. The model adapted with Mozilla degraded at the same rate as with the *update pseudo* method, while INHOUSE degraded even sooner than in the previous experiment.

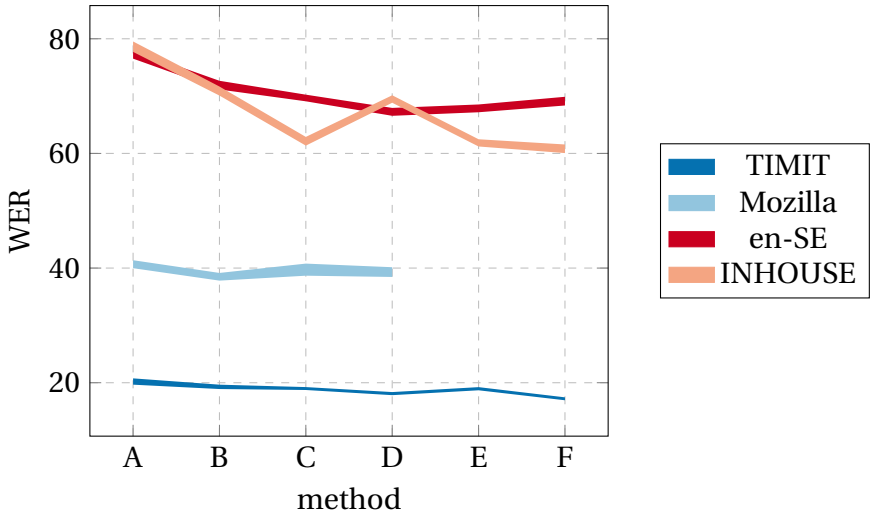

Figure 22. Evaluation results of different SSL methods. The width of the lines represents confidence intervals for WER in each method (A: baseline, B: keep pseudo, C: update pseudo, D: update pseudo, adapt baseline, E: update pseudo, ground, F: update pseudo, adapt baseline, ground).

6.3.3. Results for Additional Labeled Data

In the two remaining experiments presented in Figure 21, we used additional data randomly selected from the Mozilla dataset. We used 5 000 recordings, which were treated as manually labeled during the adaptation (no pseudo-labels generation). We decided to skip such experiments for the Mozilla dataset itself since the previous experiments (*update pseudo* and *update pseudo, adapt baseline*) gave inferior results while being very time- and resource-consuming.

Joining the idea of mixing unlabeled and labeled data with updating pseudo-labels (*update pseudo, ground*) yielded better results than only updating pseudo-labels. However, combining all three modifications: *update pseudo, adapt baseline* and *ground* gave the best results for TIMIT and INHOUSE datasets. The models performed at their best in such a configuration, and the training was stable even for 50 epochs.

A summary of the experiments described in Sections 6.3.1–6.3.3 can be found in Figure 22. The width of the lines in this figure represents confidence intervals for WER in each method. For each SSL method and dataset, checkpoints from all the epochs were compared, and the model with the lowest WER was chosen.

Table 11. WER results (in percentages) for learning with pre-training.

dataset	baseline model	pre-train	pre-train, update pseudo, adapt baseline, ground
en-SE	77.42 ± 0.89	37.82 ± 0.52	**33.71 ± 0.67**
INHOUSE	78.67 ± 0.96	59.58 ± 1.12	**47.86 ± 1.41**

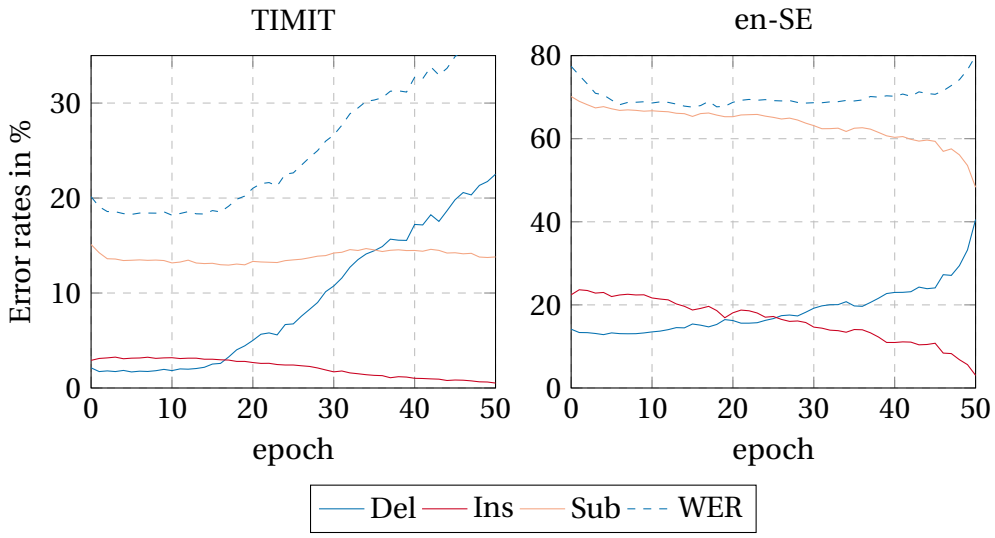

Figure 23. TIMIT and en-SE datasets detailed results (update pseudo, adapt baseline experiments).

6.3.4. Results for Pre-training with Labeled Data

In the last improvement, a small amount of labeled data from a similar dataset as the unlabeled data was used for pre-training. We used only 100 randomly chosen recordings. In the case of Mozilla, such a small amount of data did not change the model performance at all, while for TIMIT-train, the dataset was too small to extract additional recordings for the pre-training. Hence, we decided to perform SSL adaptations with en-SE and INHOUSE only. The pre-training phase was followed by *update pseudo, adapt baseline* and *ground*. Results of those adaptations can be found in Table 11. Combining pre-training with minimal amounts of data and SSL adaptations with limited datasets allowed for a significant reduction of WER from 77.42 % to 33.71 % for en-SE and from 78.67 % to 47.86 % for INHOUSE.

6.4. Discussion

In our experiments, we were able to reduce WER 2 %–17 % relative, depending on the testset. The studies presented in Section 2.3 used different experimental set-ups; hence, comparison in terms of WER reduction would not be appropriate. However, it is noteworthy that, in contrast to the aforementioned studies, we used much smaller unlabeled datasets. The datasets used in our experiments contained 4 h–70 h of audio data, while the smallest datasets used in the previous studies contained much more than 100 h of data. Hence, the adaptation process was much faster and can be applied in an on-device scenario.

We investigated separate components of WER. In Figure 23, we analyzed the results from two experiments with the TIMIT and en-SE datasets, performed using pseudo-labels updating and adaptation of the baseline model in each epoch. We observed that *Del*

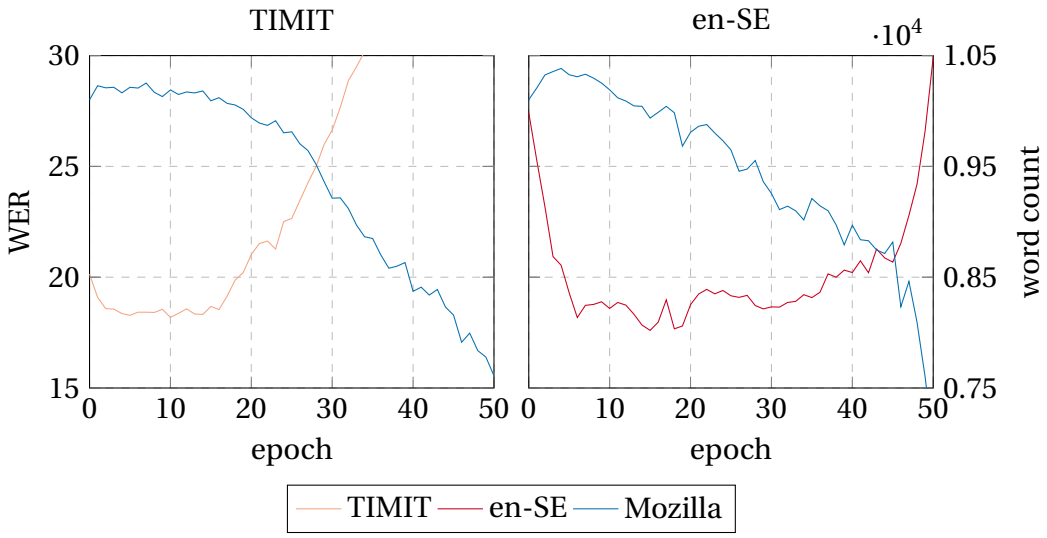

Figure 24. TIMIT and en-SE experiments (update pseudo, adapt baseline): comparison of TIMIT and en-SE testset WER and the total number of words returned by the models for Mozilla testset.

had the highest impact on the model performance. In the majority of cases, this value dropped in the initial epochs. This was followed by a small number of stable epochs. However, eventually, *Del* started to rise. On the other hand, *Sub* and *Ins* tended to decrease slowly throughout the entire adaptation process. This means that the more the model was adapted to the pseudo-labels, the more it tended to omit tokens.

The observation that *Del* is the most varying value in this type of training has led us to search for the method of epoch number optimization. One obvious solution would be to employ a manually transcribed testset with a token distribution similar to the unlabeled dataset. This idea was used to estimate the model's performance in Figure 22 and Table 11. However, this solution requires human intervention; therefore, we decided to search for a heuristic that would eliminate this issue. We checked the total number of tokens returned by the model for a given test set. As it turned out, even for test sets that represented a distribution different from the unlabeled data, this number was linked to the model performance (Figure 24). The models adapted with TIMIT and en-SE datasets (*update pseudo, adapt baseline* method) were evaluated with a random sample of 1000 utterances extracted from Mozilla. The total number of words returned by the ASR models for this testset was computed in each epoch. We also present WER in each epoch for reference. Adaptation with the TIMIT dataset gave the best results between epochs 2 and 16, with the minimum in epoch 10. The model produced the best results for en-SE between epochs 14 and 21, with the minimum in epoch 15. The number of words returned for the Mozilla testset also started dropping significantly after epoch 16 (for TIMIT) and epoch 20 (for en-SE). Therefore, we think a heuristic method can be proposed based on the total number of tokens returned by the ASR model.

As expected, our experiments confirm a significant difference in the performance

improvement that can be achieved using the same adaptation method with different datasets. We observed the great difficulty in gaining even minor improvements in the case of the Mozilla samples. We suspect this was caused by the fact that this dataset contained a wide range of vocabulary. The number of unique sentences was almost the same as the total number of recordings (see Table 10). On the other end was INHOUSE, where by using only unlabeled data and a small amount of labeled data from MCV, we reduced WER from 78.67 % to 61.06 %. INHOUSE contained a relatively small number of unique words repeated in numerous recordings. Hence, the model had a chance to learn the correct labeling for many of those words.

6.5. Conclusions

In this section, I have demonstrated that SSL methods can improve ASR performance even for very small datasets and also for data with token distributions significantly different from those of the data used to train the baseline model. I analyzed a couple of variants of the SSL algorithms in terms of WER and also detailed values of this metric's components. I proposed using a combination of *update pseudo, adapt baseline* and *ground* methods, which allowed for a reduction of WER by 2 %–17 % relative, depending on the dataset. I have also shown that the characteristic of the unlabeled dataset has a major impact on the gain caused by SSL. Finally, I have proposed a heuristic that could be used as a stopping criterion during evaluation.

7. Summary

This dissertation proposed improvements in speech processing modules in the realm of on-device use cases. The following theses were formulated and experimentally proved:

THESIS 1: The performance and accuracy of a keyword spotting model can be significantly improved by using a unigram language model and audio recordings generated by a text-to-speech system.

The key to proving this statement was a careful choice of weights. Evaluations using MOCKS showed that the EER was reduced by 13 %–17 % relative depending on the subset. Additionally, with GSC, the accuracy increased from approximately 84 % to 96 %. The value of $boost$ can be treated as the model's hyperparameter, and its optimal value depends on the testset. However, 32 seems a good choice, yielding good results for most test cases.

Introducing the multi-keyword classifier gave less impressive improvements than the unigram LM. However, its utility was visible in evaluations with GSC with the increase of accuracy from 95.97 % to 96.75 %.

It is noteworthy that the proposed improvements can be used independently and with different types of models. Unigram LMs can be applied to rescore outputs of any kind of AM, which uses the beam search algorithm to choose the most probable hypothesis. AM generating textual hypothesis can be considered an embedding model for audio data. The keyword classifier used in our solution can be extended to any embedding type by applying different distance functions. Thus, the multi-keyword approach can be applied to any solution based on a binary classifier comparing audio and text data embeddings.

THESIS 2: The accuracy of an end-of-speech detection model can be effectively improved by training the model with the proposed loss function.

The validity of this statement also depends heavily on the choice of the weights. The proposed weighting schema was evaluated with two distinct types of testsets: TIMIT and INHOUSE, with the latter being more demanding and close to the actual production audio data. The weight function proved to increase the *Fine EOS* metric by more than 27 % relative on average, with a maximum 35 % relative in INHOUSE clean (an improvement from 25 % to 60 %). It should also be noted that those improvements in *Fine EOS* were gained by reducing the *Early EOS* metric (decreased 33 % relative on average) and only a slight increase in *Fail EOS* (5 % relative on average).

The proposed loss function improvement can be applied to various model architectures. Binary cross-entropy is a standard function in this task, and extending it with additional weights is straightforward. Hence, the proposed solution can benefit any model trained with this loss function.

THESIS 3: Semi-supervised learning methods can be effectively used to adapt acoustic models even with small datasets.

Throughout the work on this statement, the goal was to adapt a generic model to significantly different data from the original training data. Even the simplest SSL method

based on fixed pseudo-labels proved this thesis. However, the gains were minor concerning the WER metric. Adding simple improvements to the primary SSL method expanded gains in WER even further. The most significant drop in this metric was observed after performing a short adaptation with a small amount of manually transcribed data before the modified SSL method. With such an approach, it was possible to lower WER from approximately 77 % to 33 %–47 %, depending on the testset. SSL is model invariant by design; hence, it can be used with different architectures.

All the research activities described in this dissertation were performed within projects at Samsung R&D Institute Poland. A prototype supported proving each of the abovementioned theses. QbyT KWS solution is still under development with the plan to further improve the results. The EOS model trained with the proposed weighted loss function is successfully used to prepare the data for tasks such as KWS or ASR. The EOS model is beneficial when audio data is provided as one long stream, but the training or evaluating procedure requires separate sentences. Finally, the methods developed for SSL are implemented for AM adaptation to new domains with the usage data. This removes the requirement to manually transcribe the data, which reduces the overall model production cost.

Many speech-processing modules are already deployed to end-user devices. However, it is just the beginning of this migration. The most computationally demanding modules, such as AMs, are still designed for the client-server architecture. Deployment of AMs to mobile devices can be supported by improving any of the modules used in speech processing pipelines. The improvements described in this thesis are good examples to support this claim:

- The lower the FPR in the KWS model, the smaller the amount of data processed by the AM, and the smaller the load of the device's resources. There is also another side of this picture, which is not connected directly with the AM: the higher the TPR in the KWS model, the higher the user experience becomes.
- Improving the precision of the EOS model can facilitate generating correct hypotheses by the AM and lower the amount of data to be processed.
- Developing methods for AM adaptation with limited unlabelled data can also allow for more aggressive compression of the model.

I firmly believe that moving multiple components of AI-based applications to mobile devices is the future in this field of science and technology. I also think the improvements presented in this thesis will be a valuable contribution to foster future work on on-device use cases of speech processing.

The results presented in this thesis open multiple interesting research problems, such as:

- What are the other unigram LM weight initialization strategies in the solution proposed for QbyT KWS?

- Does using bigram LM improve the results in the solution proposed for QbyT KWS?
- What is the impact of the multi-keyword classifier proposed for QbyT KWS if it was used with different embedding types?
- What are the other strategies for weight initialization in the proposed loss function for EOS model training?
- What is the impact of the proposed loss function for EOS model training on different model architectures?
- What is the characteristic of the dataset that works well with SSL and AM adaptation?
- How good is the heuristic based on the number of returned tokens when applied as the stopping criterion for SSL and AM adaptations?

Addressing these questions might provide further improvements in the topics which were considered in this thesis. Thus, the performance of the mobile applications based on ASR would also improve.

A. Authors' Scientific Achievements

This appendix lists all peer-reviewed articles and patents authored or co-authored by the author of this thesis.

International conferences

- **M. Pudo**, A. Wiśniewski, and A. Janicki, "Improved weighted loss function for training end-of-speech detection models", in *Proc. 18th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2020)*, 2020, Chiang Mai, Thailand [93], **Contribution**: co-design of the model training method improvement, model training and evaluation, results analysis, project supervision, **CORE B, 70 MEiN points**,
- **M. Pudo**, N. Szczepanek, B. Lukasiak, and A. Janicki, "Semi-supervised learning with limited data for automatic speech recognition", in *Proc. IEEE 7th Forum on Research and Technologies for Society and Industry Innovation (RTSI 2022)*, 2022, Paris, France [94], **Contribution**: design of the model training methods, implementation of the training scripts, model training and evaluation, results analysis, project supervision,
- K. Gabor-Siatkowska, M. Sowański, **M. Pudo**, R. Rzatkiewicz, I. Stefaniak, M. Kozłowski, and A. Janicki, "Therapeutic spoken dialogue system in clinical settings: Initial experiments", in *Proc. 30th International Conference on Systems, Signals and Image Processing, (IWSSIP 2023)*, 2023, Ohrid, North Macedonia [115], **Contribution**: research on the ASR methods in speech disorders, **20 MEiN points**,
- **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*, 2023, Dublin, Ireland [92], **Contribution**: co-design of the testset preparation procedure, baseline model training, and evaluation, project supervision, **CORE A, 140 MEiN points**.
- **M. Pudo**, M. Wosik, and A. Janicki, "Open vocabulary keyword spotting with small-footprint ASR-based architecture and language models", in *18th Conference on Computer Science and Intelligence Systems (FedCSIS 2023)*, 2023, Warsaw, Poland [91], **Contribution**: design of the LM initialization procedure, model evaluation, results analysis, project supervision, **CORE B, 20 MEiN points**.

Domestic conferences

- **M. Pudo**, N. Szczepanek, M. Sowański, B. Łukasiak, and A. Janicki, "Metody uczenia częściowo nadzorowanego w automatycznym rozpoznawaniu mowy", Poster at *XV Kopernikańskie Seminarium Doktoranckie*, 2022, Toruń, Poland [116], **Contribution**: design of the model training methods, preparation of the training scripts, model training, and evaluation, results analysis, project supervision,

- M. Sowański, **M. Pudo**, and A. Janicki, "Wykrywanie nieprzetłumaczalnych fraz w tekstach naukowych z dziedziny chemii, biologii i fizyki", Poster at *XV Kopernikańskie Seminarium Doktoranckie*, 2022, Toruń, Poland [117], **Contribution**: research on the ASR methods in the context of rare and out of vocabulary phrases.

Chapters in monographs

- **M. Pudo**, N. Szczepanek, M. Sowański, B. Łukasiak, and A. Janicki, "Metody uczenia częściowo nadzorowanego w automatycznym rozpoznawaniu mowy", in *Kopernikańskie Seminarium Doktoranckie. Na pograniczu chemii, biologii i fizyki – rozwój nauk*, vol. 4, 2022 [116], **Contribution**: confront Domestic conferences subsection, **20 MEiN points**,
- M. Sowański, **M. Pudo**, and A. Janicki, "Wykrywanie nieprzetłumaczalnych fraz w tekstach naukowych z dziedziny chemii, biologii i fizyki", in *Kopernikańskie Seminarium Doktoranckie. Na pograniczu chemii, biologii i fizyki – rozwój nauk*, vol. 4, 2022 [117], **Contribution**: confront Domestic conferences subsection, **20 MEiN points**.

Patents

- G. Kaplita, J. M. Kaczyński, M. T. Musik, and **M. Pudo**, "Method and apparatus for automated dispatch of mobile devices in a communication system", U.S. Patent 10 194 485, Jan. 29, 2019 [118] **Contribution**: co-design of the patented method, research on the state of the art.

Awards

- The 2023 Professor Zdzisław Pawlak Award in the category Industry Cooperation Award for the paper **M. Pudo**, M. Wosik, and A. Janicki, "Open vocabulary keyword spotting with small-footprint ASR-based architecture and language models", in *18th Conference on Computer Science and Intelligence Systems (FedCSIS 2023)*, 2023, Warsaw, Poland [91].

B. Custom Keyword Spotting, Additional Figures

Figure 25 presents DET curves for different subsets of MOCKS. Each curve was generated by all samples from "positive", "similar", and "different" test cases in the given subset. Even though the model was trained solely with English speech data, the DET curves for *en_LS_clean*, *en_LS_other*, and *en_MCV* were similar to *de_MCV* and *fr_MCV*. The DET curves for *es_MCV* and *it_MCV* show that the baseline model's performance was slightly worse for those languages.

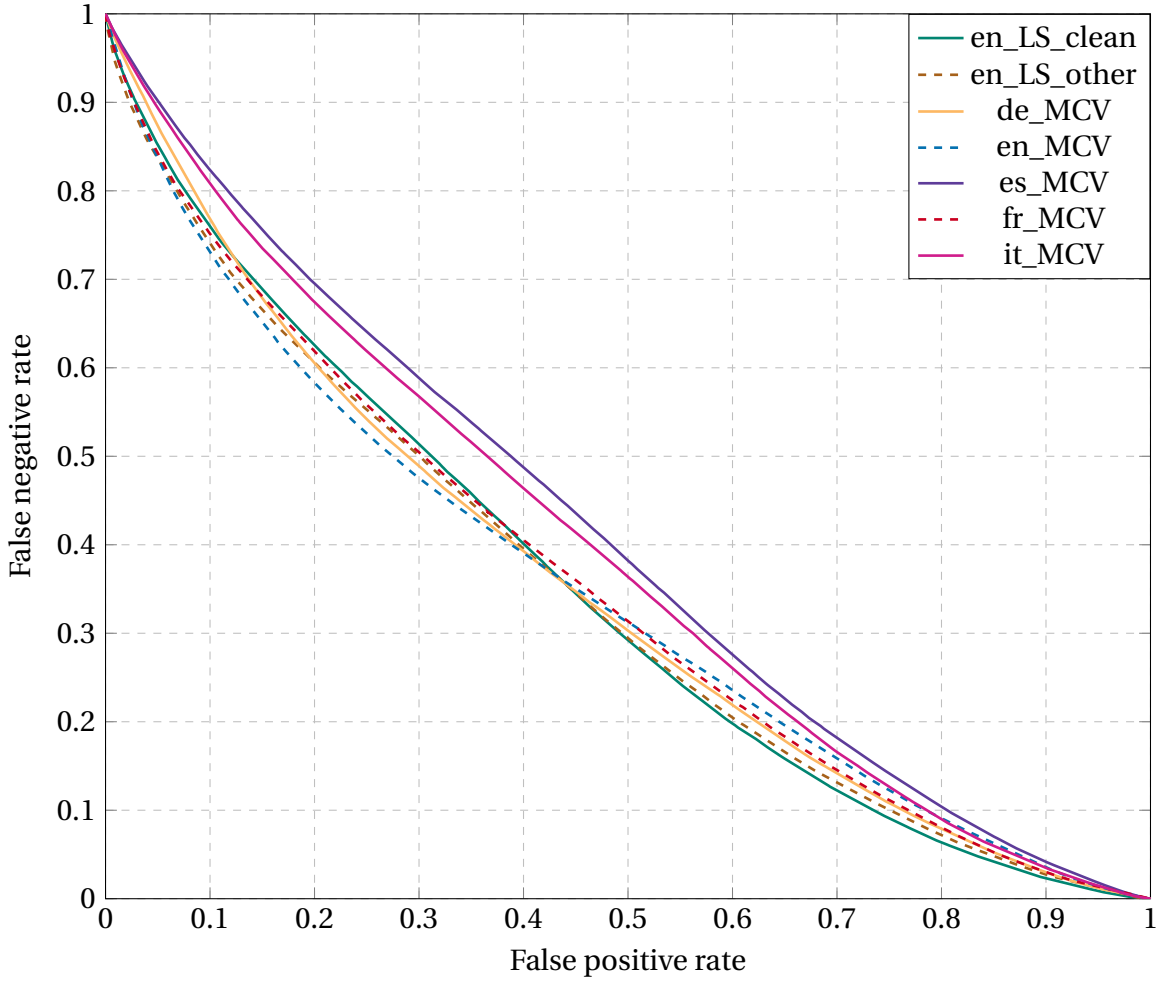

Figure 25. DET curves for different MOCKS subsets.

We also analyzed the DET curves with negative samples taken separately from "similar" and "different" subsets. Figure 26 presents the results of such analysis. As could be expected, test cases in the "similar" subset are more challenging than "different". The distances between both curves are different for all subsets in MOCKS.

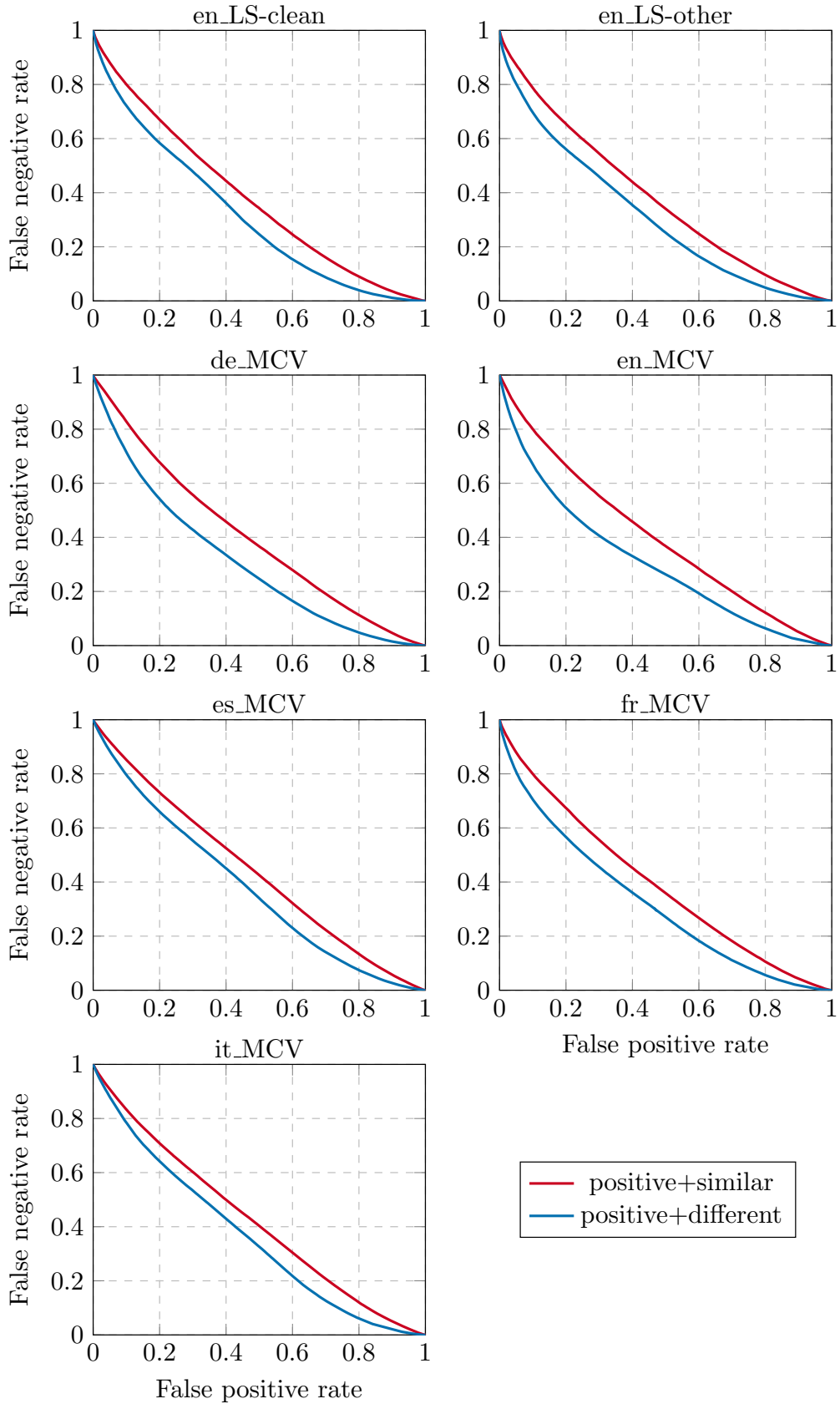

Figure 26. DET curves for different MOCKS subsets split between “similar” and “different” subsets.

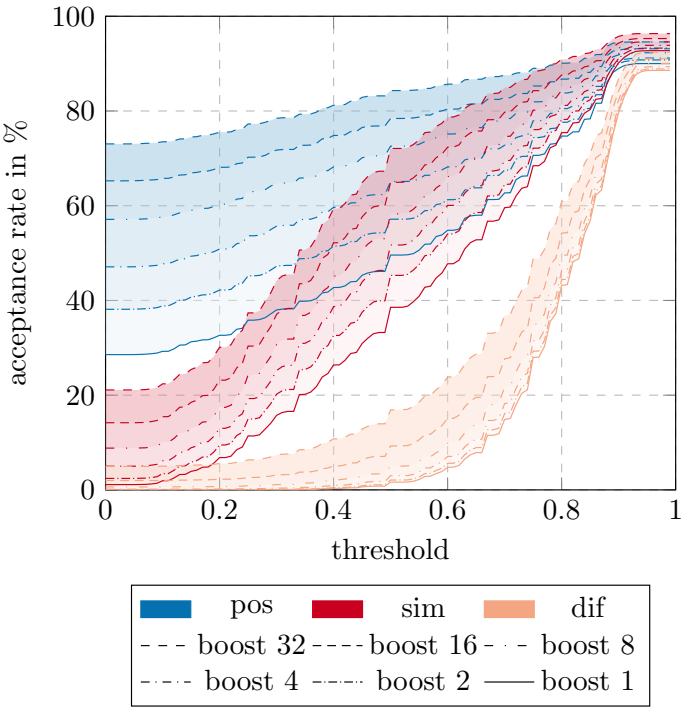

Figure 27. Boosting results with static LM, without multi-keyword classifier, for en_LS_other testset.

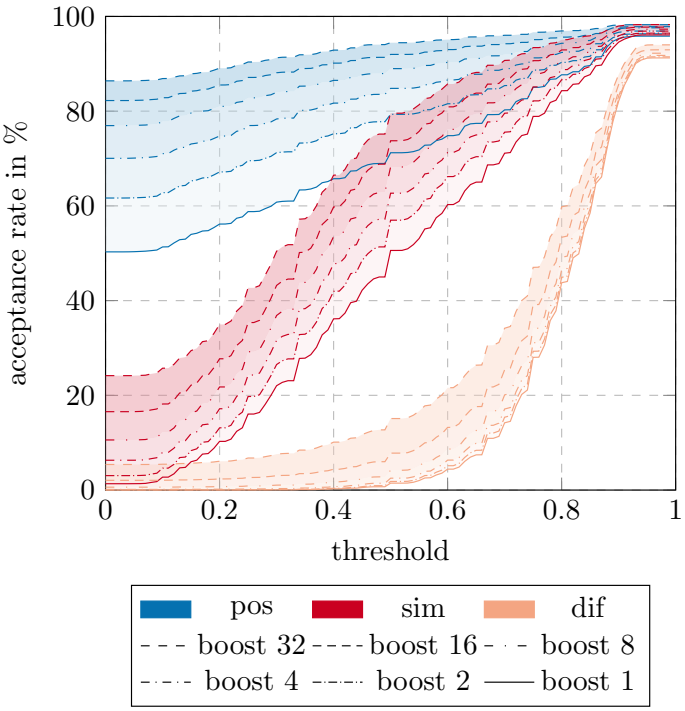

Figure 28. Boosting results with static LM, without multi-keyword classifier, for en_MCV testset.

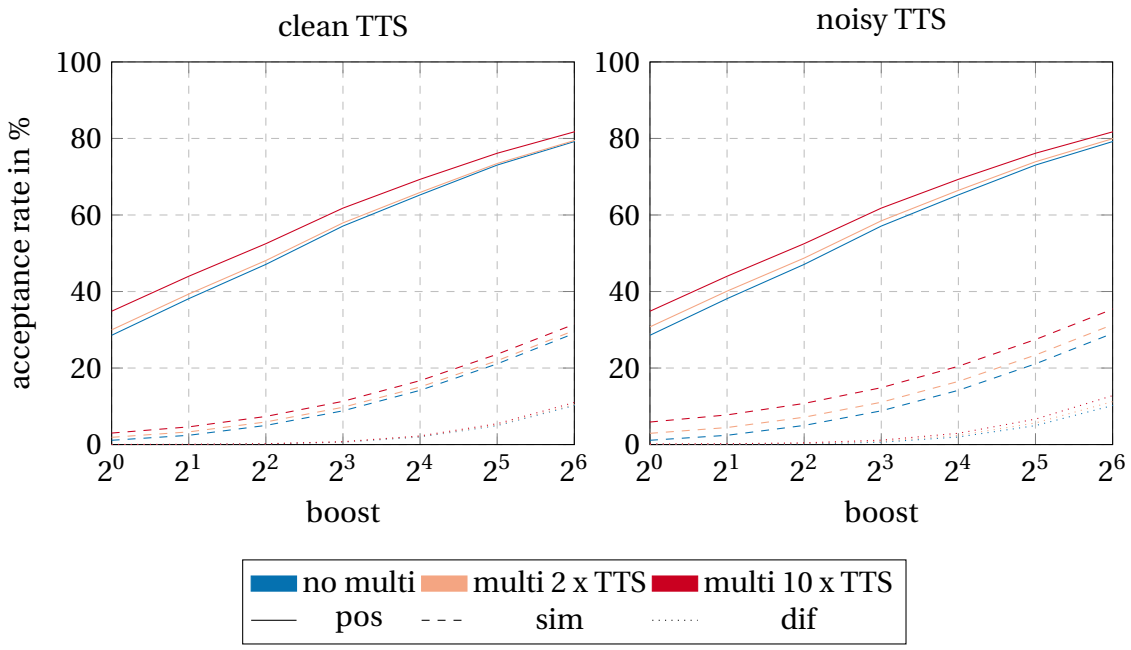

Figure 29. Boosting results with static LM, with and without multi-keyword classifier, for en_LS_other testset.

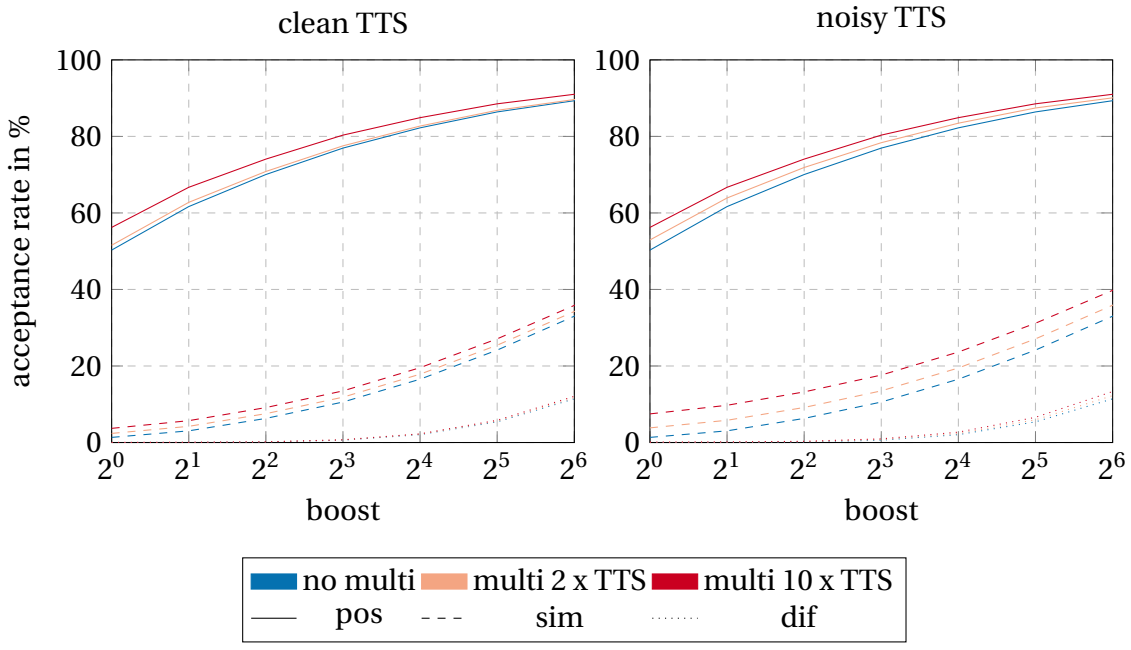

Figure 30. Boosting results with static LM, with and without multi-keyword classifier, for en_MCV testset.

References

[1] *Apple 1*, https://en.wikipedia.org/wiki/Apple_I, [Online; accessed 19-August-2023], 2023.

[2] K. Oka, Y. Sato, and H. Koike, "Real-time tracking of multiple fingertips and gesture recognition for augmented desk interface systems", in *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, IEEE, 2002, pp. 429–434.

[3] N. Gupta, P. Mittal, S. D. Roy, S. Chaudhury, and S. Banerjee, "Developing a gesture-based interface", *IETE Journal of Research*, vol. 48, no. 3-4, pp. 237–244, 2002.

[4] A. E. Kaufman, A. Bandopadhay, and B. D. Shaviv, "An eye tracking computer user interface", in *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*, IEEE, 1993, pp. 120–121.

[5] C. H. Morimoto and M. R. Mimica, "Eye gaze tracking techniques for interactive applications", *Computer vision and image understanding*, vol. 98, no. 1, pp. 4–24, 2005.

[6] B. Schmidt, R. Borrison, A. Cohen, *et al.*, "Industrial virtual assistants: Challenges and opportunities", in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, 2018, pp. 794–801.

[7] A. S. Tulshan and S. N. Dhage, "Survey on virtual assistant: Google assistant, siri, cortana, alexa", in *Advances in Signal Processing and Intelligent Recognition Systems: 4th International Symposium SIRS 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers 4*, Springer, 2019, pp. 190–201.

[8] C. S. Babu, M. Devasya, V. Sudeep, S. Kumar, and C. K. Mandal, "Interactive voice response for the visually challenged", in *Investigations in Pattern Recognition and Computer Vision for Industry 4.0*, IGI Global, 2023, pp. 180–196.

[9] M. R. Janevic, A. C. Aruquipa Yujra, N. Marinec, *et al.*, "Feasibility of an interactive voice response system for monitoring depressive symptoms in a lower-middle income latin american country", *International journal of mental health systems*, vol. 10, pp. 1–11, 2016.

[10] R. Zhang, W. Chen, M. Xu, and Y. Yang, "Analysis and design of voice assisted learning system based on baidu ai", in *2019 IEEE International Conference on Computer Science and Educational Informatization (CSEI)*, IEEE, 2019, pp. 334–336.

[11] N. Isaila and I. Nicolau, "Promoting computer assisted learning for persons with disabilities", *Procedia-Social and Behavioral Sciences*, vol. 2, no. 2, pp. 4497–4501, 2010.

[12] A. Ilievski, D. Dojchinovski, and M. Gusev, "Interactive voice assisted home healthcare systems", in *Proceedings of the 9th Balkan Conference on Informatics*, 2019, pp. 1–5.

[13] R. Chopda, A. Khan, A. Goenka, D. Dhere, and S. Gupta, "An intelligent voice assistant engineered to assist the visually impaired", in *Intelligent Computing and Networking: Proceedings of IC-ICN 2022*, Springer, 2023, pp. 143–155.

[14] R. Garg, P. Kumar, S. Tomar, V. Deep, and D. Mehrotra, "Voice assisted smart home automation system", 2021.

[15] B Bharathi and S. Chatterjee, "A cost effective implementation of a voice assisted home automation system", *Applied Mechanics and Materials*, vol. 704, pp. 390–394, 2015.

[16] R. Cutler, A. Saabas, T. Parnamaa, *et al.*, "Icassp 2022 acoustic echo cancellation challenge", in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2022, pp. 9107–9111.

[17] G. M. Davis, *Noise reduction in speech applications*. CRC press, 2018.

[18] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, "Automatic speech recognition: A survey", *Multimedia Tools and Applications*, vol. 80, pp. 9411–9457, 2021.

[19] M. Sunkara, C. Shivade, S. Bodapati, and K. Kirchhoff, "Neural inverse text normalization", in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 7573–7577.

[20] 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.

[21] 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, 627–630 vol.1. DOI: 10.1109/ICASSP.1989.266505.

[22] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques", in *ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, 1991, 309–312 vol.1. DOI: 10.1109/ICASSP.1991.150338.

[23] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

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

[25] S. Panchapagesan, M. Sun, A. Khare, *et al.*, "Multi-Task Learning and Weighted Cross-Entropy for DNN-Based Keyword Spotting", in *Proc. Interspeech 2016*, 2016, pp. 760–764. DOI: 10.21437/Interspeech.2016-1485.

[26] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks", in *ICML 2006 - Proceedings of the 23rd International Conference on Machine Learning*, vol. 2006, Jan. 2006, pp. 369–376. DOI: 10.1145/1143844.1143891.

[27] 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.

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

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

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

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

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

[33] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword Transformer: A Self-Attention Model for Keyword Spotting", in *Proc. Interspeech 2021*, 2021, pp. 4249–4253. DOI: 10.21437/Interspeech.2021-1286.

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

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

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

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

[38] 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 *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 6858–6862. DOI: 10.1109/ICASSP39728.2021.9414156.

[39] 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, pp. 5200–5204. DOI: 10.21437/Interspeech.2022-11080.

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

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

[42] P. Warden, *Speech commands: A dataset for limited-vocabulary speech recognition*, 2018. DOI: 10.48550/ARXIV.1804.03209. [Online]. Available: https://arxiv.org/abs/1804.03209.

[43] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted Residual Learning for Efficient Keyword Spotting", in *Proc. Interspeech 2021*, 2021, pp. 4538–4542. DOI: 10.21437/Interspeech.2021-383.

[44] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming Keyword Spotting on Mobile Devices", in *Proc. Interspeech 2020*, 2020, pp. 2277–2281. DOI: 10.21437/Interspeech.2020-1003.

[45] R. Tang, W. Wang, Z. Tu, and J. Lin, "An experimental analysis of the power consumption of convolutional neural networks for keyword spotting", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 5479–5483.

[46] I. López-Espejo, Z.-H. Tan, and J. Jensen, "Exploring filterbank learning for keyword spotting", in *2020 28th European Signal Processing Conference (EUSIPCO)*, IEEE, 2021, pp. 331–335.

[47] I. López-Espejo, Z.-H. Tan, and J. Jensen, "A novel loss function and training strategy for noise-robust keyword spotting", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 2254–2266, 2021. DOI: 10.1109/TASLP.2021.3092567.

[48] M. Mazumder, S. Chitlangia, C. Banbury, *et al.*, "Multilingual spoken words corpus", in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[49] R. Ardila, M. Branson, K. Davis, *et al.*, "Common voice: A massively-multilingual speech corpus", in *International Conference on Language Resources and Evaluation*, 2019.

[50] 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, pp. 773–785.

[51] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger, "Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi", in *Proc. Interspeech 2017*, 2017, pp. 498–502. DOI: 10.21437/Interspeech.2017-1386.

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

[53] D. Hossain and Y. Sato, "Efficient corpus design for wake-word detection", in *2021 IEEE Spoken Language Technology Workshop (SLT)*, 2021, pp. 1094–1100. DOI: 10.1109/SLT48900.2021.9383569.

[54] A. Larcher, K. A. Lee, B. Ma, and H. Li, "Text-dependent speaker verification: Classifiers, databases and rsr2015", *Speech Communication*, vol. 60, pp. 56–77, 2014.

[55] J. Garofolo, L. Lamel, W. Fisher, *et al.*, *TIMIT acoustic-phonetic continuous speech corpus*, Linguistic Data Consortium, Nov. 1993.

[56] X. Qin, H. Bu, and M. Li, "Hi-mia: A far-field text-dependent speaker verification database and the baselines", in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2020, pp. 7609–7613.

[57] J. Hou, Y. Shi, M. Ostendorf, M.-Y. Hwang, and L. Xie, "Region proposal network based small-footprint keyword spotting", *IEEE Signal Processing Letters*, vol. 26, no. 10, pp. 1471–1475, 2019.

[58] R. Hariharan, J. Hakkinen, and K. Laurila, "Robust end-of-utterance detection for real-time speech recognition applications", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, vol. 1, Salt Lake City, UT, USA: IEEE, Jan. 2001, pp. 249–252, ISBN: 0-7803-7041-4. DOI: 10.1109/ICASSP.2001.940814.

[59] H. Arsikere, E. Shriberg, and U. Ozertem, "Computationally-efficient endpointing features for natural spoken interaction with personal-assistant systems", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014)*, Florence, Italy: IEEE, May 2014, pp. 3241–3245, ISBN: 978-1-4799-2893-4. DOI: 10.1109/ICASSP.2014.6854199.

[60] H. Arsikere, E. Shriberg, and U. Ozertem, "Enhanced end-of-turn detection for speech to a personal assistant", in *AAAI Spring Symposium - Technical Report*, vol. SS-15-07, Palo Alto, CA, USA: AAAI, 2015, pp. 75–78, ISBN: 9781577357117.

[61] L. Ferrer, E. Shriberg, and A. Stolcke, "A prosody-based approach to end-of-utterance detection that does not require speech recognition", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2003)*, Hong Kong, China: IEEE, May 2003, pp. I–608. DOI: 10.1109/ICASSP.2003.1198854.

[62] C. Xu, L. Xie, G. Huang, X. Xiao, E. S. Chng, and H. Li, "A deep neural network approach for sentence boundary detection in broadcast news", in *Fifteenth annual conference of the international speech communication association*, Citeseer, 2014.

[63] M. Hasan, R. Doddipatla, and T. Hain, "Multi-pass sentence-end detection of lecture speech", in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[64] M. Shannon, G. Simko, S.-Y. Chang, and C. Parada, "Improved end-of-query detection for streaming speech recognition", in *Proc. INTERSPEECH 2017*, Stockholm, Sweden: ISCA, 2017, pp. 1909–1913.

[65] A. Mohamed, H.-y. Lee, L. Borgholt, *et al.*, "Self-supervised speech representation learning: A review", *arXiv preprint arXiv:2205.10643*, 2022.

[66] R. Balestriero, M. Ibrahim, V. Sobal, *et al.*, "A cookbook of self-supervised learning", *arXiv preprint arXiv:2304.12210*, 2023.

[67] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "Wav2vec: Unsupervised pre-training for speech recognition", *arXiv preprint arXiv:1904.05862*, 2019.

[68] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "Wav2vec 2.0: A framework for self-supervised learning of speech representations", *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[69] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision", in *International Conference on Machine Learning*, PMLR, 2023, pp. 28 492–28 518.

[70] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning", *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.

[71] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning", *arXiv preprint arXiv:2006.05278*, 2020.

[72] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning", *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[73] G.-J. Qi and J. Luo, "Small data challenges in big data era: A survey of recent progress on unsupervised and semi-supervised methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[74] A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. Goodfellow, "Realistic evaluation of deep semi-supervised learning algorithms", *Advances in neural information processing systems*, vol. 31, 2018.

[75] P. Bachman, O. Alsharif, and D. Precup, "Learning with pseudo-ensembles", *Advances in neural information processing systems*, vol. 27, 2014.

[76] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning", *arXiv preprint arXiv:1610.02242*, 2016.

[77] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning", *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 1979–1993, 2018.

[78] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization", *Advances in neural information processing systems*, vol. 17, 2004.

[79] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks", in *Workshop on challenges in representation learning, ICML*, Atlanta, vol. 3, 2013, p. 896.

[80] L. Chen and V. Leutnant, "Acoustic model bootstrapping using semi-supervised learning.", in *Proc. Interspeech*, 2019, pp. 3198–3202.

[81] D. S. Park, Y. Zhang, Y. Jia, *et al.*, "Improved noisy student training for automatic speech recognition", *arXiv:2005.09629*, 2020.

[82] Y. Zhang, J. Qin, D. S. Park, *et al.*, "Pushing the limits of semi-supervised learning for automatic speech recognition", *ArXiv:2010.10504*, 2020.

[83] G. Synnaeve, Q. Xu, J. Kahn, *et al.*, "End-to-end ASR: From supervised to semi-supervised learning with modern architectures", *arXiv:1911.08460*, 2019.

[84] J. Kahn, M. Riviere, W. Zheng, *et al.*, "Libri-light: A benchmark for ASR with limited or no supervision", in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, IEEE, 2020. DOI: 10.1109/icassp40776.2020.9052942.

[85] Q. Xu, T. Likhomanenko, J. Kahn, A. Hannun, G. Synnaeve, and R. Collobert, "Iterative pseudo-labeling for speech recognition", *arXiv preprint arXiv:2005.09267*, 2020.

[86] J. Kearns, "Librivox: Free public domain audiobooks", *Reference Reviews*, vol. 28, no. 1, pp. 7–8, 2014.

[87] Y. Higuchi, N. Moritz, J. L. Roux, and T. Hori, "Momentum pseudo-labeling for semi-supervised speech recognition", in *Proc. Interspeech 2021*, 2021. arXiv: 2106.08922 [eess.AS].

[88] F. Hernandez, V. Nguyen, S. Ghannay, N. Tomashenko, and Y. Estève, "Ted-lium 3: Twice as much data and corpus repartition for experiments on speaker adaptation", in *Speech and Computer*, Springer International Publishing, 2018, pp. 198–208.

[89] L. Lugosch, T. Likhomanenko, G. Synnaeve, and R. Collobert, "Pseudo-labeling for massively multilingual speech recognition", *arXiv:2111.00161*, 2021.

[90] E. Wallington, B. Kershenbaum, O. Klejch, and P. Bell, "On the learning dynamics of semi-supervised training for asr", in *Proc. Interspeech 2021*, 2021, pp. 716–720. DOI: 10.21437/Interspeech.2021-1777.

[91] M. Pudo, M. Wosik, and A. Janicki, "Open vocabulary keyword spotting with small-footprint asr-based architecture and language models", in *18th Conference on Computer Science and Intelligence Systems (FedCSIS 2023)*, 2023.

[92] M. Pudo, M. Wosik, A. Cieślak, J. Krzywdziak, B. Lukasiak, and A. Janicki, "MOCKS 1.0: Multilingual Open Custom Keyword Spotting Testset", in *Proc. INTERSPEECH 2023*, 2023, pp. 4054–4058. DOI: 10.21437/Interspeech.2023-1049.

[93] M. Pudo, A. Wiśniewski, and A. Janicki, "Improved weighted loss function for training end-of-speech detection models", in *Proc. 18th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2020)*, Chiang Mai, Thailand: ACM, 2020, 25–29, ISBN: 9781450389242. DOI: 10.1145/3428690.3429160. [Online]. Available: https://doi.org/10.1145/3428690.3429160.

[94] M. Pudo, N. Szczepanek, B. Lukasiak, and A. Janicki, "Semi-supervised learning with limited data for automatic speech recognition", in *Proc. IEEE 7th Forum on Research and Technologies for Society and Industry Innovation (RTSI 2022)*, Paris, France: IEEE, 2022, pp. 136–141. DOI: 10.1109/RTSI55261.2022.9905112.

[95] S. Settle and K. Livescu, "Discriminative acoustic word embeddings: Recurrent neural network-based approaches", in *2016 IEEE Spoken Language Technology Workshop (SLT)*, IEEE, 2016, pp. 503–510.

[96] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition", in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2016, pp. 4960–4964.

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

[98] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey", *International Journal of Computer Vision*, vol. 129, no. 6, 1789–1819, 2021, ISSN: 0920-5691. DOI: 10.1007/s11263-021-01453-z.

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

[100] D. S. Park, W. Chan, Y. Zhang, *et al.*, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition", in *Proc. Interspeech 2019*, 2019, pp. 2613–2617. DOI: 10.21437/Interspeech.2019-2680.

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

[102] 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, Jul. 2016. DOI: 10.1587/transinf.2015EDP7457.

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

[104] J. Shen, R. Pang, R. J. Weiss, *et al.*, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4779–4783. DOI: 10.1109/ICASSP.2018.8461368.

[105] N. Ellinas, G. Vamvoukakis, K. Markopoulos, *et al.*, "High Quality Streaming Speech Synthesis with Low, Sentence-Length-Independent Latency", in *Proc. Interspeech 2020*, 2020, pp. 2022–2026. DOI: 10.21437/Interspeech.2020-2464.

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

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

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

[109] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2015)*, Brisbane, Australia: IEEE, 2015, pp. 4580–4584.

[110] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, *et al.*, "Audio set: An ontology and human-labeled dataset for audio events", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*, New Orleans, LA, USA: IEEE, 2017, pp. 776–780.

[111] J. Kahn, A. Lee, and A. Y. Hannun, "Self-training for end-to-end speech recognition", *CoRR*, vol. abs/1909.09116, 2019. arXiv: 1909.09116. [Online]. Available: http://arxiv.org/abs/1909.09116.

[112] K. Kim, K. Lee, D. Gowda, *et al.*, "Attention based on-device streaming speech recognition with large speech corpus", *arXiv:2001.00577*, 2020. eprint: 2001.00577.

[113] D. Lee, P. Kapoor, and B. Kim, "Deeptwist: Learning model compression via occasional weight distortion", *CoRR*, vol. abs/1810.12823, 2018. arXiv: 1810.12823. [Online]. Available: http://arxiv.org/abs/1810.12823.

[114] I. Demirsahin, O. Kjartansson, A. Gutkin, and C. Rivera, "Open-source Multi-speaker Corpora of the English Accents in the British Isles", in *Proc. 12th Language Resources and Evaluation Conference (LREC 2020)*, Marseille, France: European Language Resources Association (ELRA), May 2020, pp. 6532–6541.

[115] K. Gabor-Siatkowska, M. Sowański, M. Pudo, *et al.*, "Therapeutic spoken dialogue system in clinical settings: Initial experiments", in *Proc. 30th International Conference on Systems, Signals and Image Processing, (IWSSIP 2023)*, Ohrid, North Macedonia: IEEE, 2023, pp. 1–4. DOI: 10.1109/IWSSIP55020.2023.00000.

[116] M. Pudo, N. Szczepanek, M. Sowański, B. Łukasiak, and A. Janicki, "Metody uczenia częściowo nadzorowanego w automatycznym rozpoznawaniu mowy", in *Kopernikańskie Seminarium Doktoranckie. Na pograniczu chemii, biologii i fizyki – rozwój nauk*, vol. 4, Prezentacja: XV Kopernikańskie Seminarium Doktoranckie, 20-22.06.2022, Toruń, Poland, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2022.

[117] M. Sowański, M. Pudo, and A. Janicki, "Wykrywanie nieprzetłumaczalnych fraz w tekstach naukowych z dziedziny chemii, biologii i fizyki", in *Kopernikańskie Seminarium Doktoranckie. Na pograniczu chemii, biologii i fizyki – rozwój nauk*, vol. 4, Prezentacja: XV Kopernikańskie Seminarium Doktoranckie, 20-22.06.2022, Toruń, Poland, Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, 2022.

[118] G. Kaplita, J. M. Kaczyński, M. T. Musik, and M. Pudo, *Method and apparatus for automated dispatch of mobile devices in a communication system*, U.S. Patent 10 194 485, Jan. 29, 2019.