Development of a neural network library for resource constrained speech synthesis

Sujeendran Menon, Paweł Zarzycki
SmartlLife Systems Sp. z O.O.
Warsaw, Poland
S.Menon,P.Zarzycki@smartlife.global

Maria Ganzha
Warsaw University of Technology
Warsaw, Poland
M.Ganzha@mini.pw.edu.pl

Marcin Paprzycki
Systems Research Institute
Polish Academy of Sciences
Warsaw, Poland
Marcin.Paprzycki@ibspan.waw.pl

***Abstract*—Machine learning frameworks, like Tensorflow and PyTorch, use GPU hardware acceleration to deliver the needed performance. Since GPUs require a lot of power (and space) to operate, typical use cases involve high-performance servers, with the final deployment available as a cloud service. To address limitations of this approach, AI Accelerators have been proposed. In this context, we have designed and implemented a library of neural network algorithms, to efficiently run on "edge devices", with AI Accelerators. Moreover, a unified interface has been provided, to allow easy experimentation with various neural networks applied to the same dataset. Here, let us stress that we do *not* propose new algorithms, but port known ones to, resource restricted, edge devices. The context is provided by a speech synthesis application for edge devices that is deployed on an NVIDIA Jetson Nano. This application is to be used by social robots for real-time off-cloud text-to-speech processing.**

***Index Terms*—speech-synthesis, AI accelerators, optimization**

I. Introduction

The term, AI Accelerator (AIA) is, usually, designated for hardware specifically built to increase the performance of neural network (NN) models, by matching the structure of computer hardware with that of the trained/applied neural network. These systems are, typically, edge devices with less computing capabilities than the large, full-scale, GPU systems. However, they can power real-time neural models, e.g. for object detection or image classification. AIA's often materialize on small, single board, computers, to deliver cheap decentralized smart systems, which can be taken off the cloud. Since the AIA's have appeared only recently, novel applications, outside of computer vision, continue to emerge. In this context, responding to the actual needs of the SmartLife company, AIA's are being explored (their capabilities and limitations) in, among others, accurate real-time speech synthesis and recognition, and motor PID controls. Here, we are interested in scenarios where the AIA's are placed on-board of the target devices and work "autonomously". Let us note that, in this context, autonomy means that the device will not be connected to the Internet (i.e. to the cloud service). This is needed since SmartLife is developing social robots (designed to interact with humans in a natural way).

Therefore, the goal of this work is to explore ways of delivering an NN-based solution, to be placed (off-line) on a social robot, to synthesize, in real-time, natural speech, based on text inputs. In this context, existing NN algorithms were turned into a library, deployed on the NVIDIA Jetson Nano. Moreover, a unified interface for the library was developed, to allow experimenting with different NNs, applied to the same dataset, and to fine tune the NN model. Model tuning, in speech synthesis, means, among others, balancing the quality of generated speech and the speed of speech generation. Moreover, a customized API, for using the model on edge devices (in production systems), was also developed, allowing modular functions for preprocessing input text, synthesizing output, and post-processing speech for playback. Note that, since SmartLife is an SME, the developed software is its intellectual property and, at least for the time being, will not be open sourced. Therefore, material presented here is the maximum that the company decided to release at this time.

This being said, the remaining parts of this work are organized as follows. We start with a summary of the related works that were studied to understand of the evolution of state of the art in embedded speech synthesis solutions, over the past few decades. This is followed by the design considerations for the components in our proposed solution. We also provide details of the models incorporated and how they were trained, using our custom dataset, and optimized for embedded platforms. The evaluation results of our library's performance on different platforms is discussed before stating the concluding remarks and enumerating limitations of our system.

II. Related Work

Over the last 15 years, a very large amount of research has been devoted to application of NNs in speech generation and recognition. However, it can be easily noticed that most of considered models rely on the processing power provided by (large) cloud systems. Obviously, until recently, there was no need for edge-device-focused work. Only in 2019, Google launched a speech-to-text model [1] on the Gboard in the Pixel phones, which allows offline, on-device, recognition in real-time. While typical models used for this purpose require almost 450MB of space, the Google model uses only 85MB.

Keeping this in mind, let us now discuss the state-of-the-art in speech synthesis on embedded devices, and methods that make the model smaller and more efficient. Note that, while some presented results concern also speech recognition, in this work we are interested *only* in speech synthesis. Moreover, while some presented works are not focused directly on speech synthesis, they propose how NN models can be converted, or pruned, for the deployment on resource restricted devices. These proposals may consider methods that are very different from the one used in our work but, nevertheless, are of direct interest, as they have the same overarching goal in mind, i.e. how to take a large NN model and make it “smaller” to run on a limited-resource device.

In [2], Black and Lenzo reported on the design of a small and fast library, called Flite that is suitable for embedded systems and servers, generating voices based on text input. Moreover, the size and speed were given high priority. The proposed system consists of three parts, to accomplish the voice synthesis: language model, lexicon and voice. In the language model, it provides: (i) phoneset, (ii) tokenization rules, (iii) text analysis, and (iv) prosodic structures. The lexicon deals with the calibrating rules, for the vocabulary words, based on the letter to sound rule. The language segment, and the lexicon, can be expressed in the same language. However, they have separate, individual, libraries. The voice segment is the prosody model, based on a speaker, and is defined by the primitives provided by the language model. As the library is written in C++, it supports cross-platform functionality. In order to ensure the efficiency of the Flite system, voice samples, with frequency of 8KHz, were collected. However, it should be noted that these 8KHz samples result in a very low quality. Overall, Flite is a small and fast embedded system based on a library server, but does not deliver quality that would be acceptable for the intended use in social robots.

Statistical approaches, like the Hidden Markov Model (HMM), gained popularity after 1999 (see, [3], [4]). They generate similar-sounding speech clips, based on voice parameters. This is completely different from the unit selection method that used speech segments from the database. The HMM is used to generate a complex acoustic model, while (i) reducing the footprint, (ii) tuning parameters, and (iii) delivering separate control over the voice spectrum, magnitude, and duration. Separately, HMM provided techniques used in speech recognition. The drawbacks of this approach are mainly related to the acoustic modeling accuracy, and the over-smoothing of the output waveform, resulting in poor quality of speech. Even though there have been several attempts at overcome these issues, they requires careful design and tuning of each individual component parameter to ”get it right”.

The, IBM proposed, stylistic synthesis [5] was one of the first approaches to showcase variations in recordings. However, this approach requires large databases that can be costly and very difficult to create. Moreover, it does not support running speech synthesis on restricted-resource devices.

In the work, described in [6], an expressive speech, based on the Emotion Markup Language [7], and the MARY Text-to-Speech (TTS), are introduced. Here, an audiobook, built-in the HMM-based voice, segregated according to voice styles, is used to create an expressive unit selection. Principal component analysis, based on acoustic features, acquired from segmented sentences, is used to segregate the audiobook data. Note that the voicing strengths, extracted from several bands, and the voicing rate, have very good mutual relationship with expressivity, in the audiobook data.

In [8], the authors proposed Quantized Convolutional NN (CNN) that allows to simultaneously speed-up the computation and reduce memory and storage overhead of the models. They managed to quantize the filter kernels in the convolutional and fully connected layers of the CNN, while aiming at minimizing the error in each layer’s response. As a result they were able to achieve a 4-6x speed-up, and up to 20x compression, with just 1% of loss in accuracy. This work was one of the first attempts to explore reducing the precision of the models from floating 32bit to integer 16bit, or 8bit, while retaining the accuracy of the results. Nowadays, frameworks, like Tensorflow, allow post-training quantization and quantization-aware training. This allows the models to be quantized in later stages of the deployment, matching the device to be used.

In 2016, a system called Idlak Tangle [9] was proposed. It is a library, based on the open source speech recognition system. Speech is modeled using a deep neural network (DNN). It is a system, in which text processing happens in the front end. Here, the vocoder, used at the output, is based on a mixed excitation Mel log spectrum approximation (MLSA) [10]. Experimental results showed that the systems attained higher scores in the MUSHRA tests [11] and proved that it sounds more natural and performed better than the HMM-based Speech Synthesis System solutions. Based on the research reported in this paper, it can be stated that Idlak Tangle is the first DNN-based parametric synthesis system, with no usage restrictions. However, this solution is still not considered as the current state of the art as the vocoder model, used there, produces largely invariant, or monotonous, speech. While we draw inspiration from the Open Source approach of Tangle, we focus on improving the system’s capabilities, as it is neither better nor worse than the proprietary HMM-based solutions.

In the 2018 paper [12], the architecture proposed by Google in [13], was improved, by incorporating a Wavenet based vocoder for the final stage of voice synthesis, rather than the, earlier considered, Griffin-Lim method [14]. Wavenet model [15] is a fully probabilistic and autoregressive deep neural network model for generating raw audio waveforms, with the predictive distribution for each audio sample conditioned on all previous ones. The system consisted of: (a) prediction network, (b) Mel scale spectrogram, and (c) a modified Wavenet model. The prediction network compiles a recurring sequence, which is mapped to the Mel scale spectrogram. After the mapping, a modified Wavenet model is used by the vocoder. This Wavenet is trained separately, usually by feeding the model spectrograms and audio samples. This approach has been experimentally shown to outperform its predecessor. Nevertheless, even after simplification, this model is suitable *only* for large scale servers, and high powered GPUs. However, the cited paper contains an important idea of using the simpler Long short-term memory (LSTM) cells, instead of, more complex, Gated Recurrent Unit (GRU) cells, for the Recurrent NN (RNN) to simplify the model.

In summary, works summarized above proved capable of solving smaller parts of the overall problem we need to solve to have a lightweight offline speech synthesis solution for a social robot. Statistical models based on HTS and rule based systems were light enough to be portable, while DNN-based systems provided rich expressive voice, while allowing for quantization to compensate for their large sizes.

III. Designing speech generator for resource-restricted devices

Taking into account the presented background, let us now discuss the proposed approach to speech generation. First, let us summarize the existing devices, platforms and frameworks, to outline the main idea as to why each of them can be considered for the final solution.

A. Devices, platforms and frameworks

Recall that the aim of this work is to design TTS solution for social robots (their controller boards). Hence, we have analyzed Single Board Computers (SBCs) and the add-ons that they support. The most popular SBC is a Raspberry Pi. Therefore, a Raspberry Pi 3B+ that has an on-board 64-bit ARM SoC, with 1GB RAM, and multiple communication buses and interfaces, was selected for initial prototyping. Since its operating system, is a Raspbian OS, a Linux-based environment (with some unimportant deviations) was available. These devices have also Bluetooth and WiFi connectivity, as they are intended, among others, for IoT applications.

The second platform that was considered was Intel Movidius. Here, however, we have run into a number of issues with the existing, at that time, version of Open Vino framework. Since, we are told, these issues have been already solved, we only mention this as a reason why Movidius was not pursued further. At the time of our research, we could not use it.

Therefore, the Nvidia Jetson Nano platform was considered. This SBC has an onboard GPU and a newer 64-bit ARM processor, with 4GB RAM, WiFi, Bluetooth and GPIO interfaces, similar to that of the Raspberry Pi 3B+ model. It uses an Nvidia's custom version of Ubuntu 18.04 LTS with their GPU drivers, and the CUDA SDK. This makes it compatible with popular machine learning frameworks, like Tensorflow and Pytorch. Moreover, it allows training of small models onboard, with full GPU support, which is very attractive for deployment of AI models to small robots.

Mentioned Tensorflow is a, Google developed, library for machine learning. The library code is production-ready, and supports multiple model pipelines, device architectures, and development frameworks. It also provides tools to prune and optimize the model to reduce its time and memory footprint. Although the base code is in C++, it has wrapper libraries in Python, Java and Javascript. Additionally, the Tensorflow Lite, and an experimental Tensorflow Micro, help trimming down resources required for mobile devices and microcontrollers.

PyTorch is a machine learning library developed by Facebook. Comparing their most recent releases we have found that Tensorflow is more suitable for production models, and scalability. PyTorch, on the other hand, is somewhat better for rapid prototyping. Thus, we used both frameworks.

B. Speech synthesis algorithms

For implementing TTS systems, on low powered devices, selecting the right approach is crucial. There exist multiple options with different resource demand and quality level. Let us summarize approaches selected for the TTS, on the basis of experiments (details of which are out of scope, here).

1) Tacotron: Tacotron was first proposed by Google in the 2017 (see, [13]), and later improved ([16], [17]). It is a more complex version of the standard encoder-decoder structure. It consists of a custom component that consists of a convolutional bank, highway networks, and a bidirectional RNN. As noted above, in our work, in the RNN, the GRU cells have been replaced with the LSTM cells, as they have an extra gate that can help when processing longer sentences. This model relies on external vocoders to convert the spectrogram output, of the encoder-decoder, to the waveform.

2) Deep Voice 3: Deep Voice 3 was proposed by Baidu in 2017 [18]. It is also encoder-decoder-based, but relies on CNNs (without RNN components). It is one of the fastest available models. Compared to all available, at the time of our work, TTS systems, Deep Voice 3 was the only one with a comparable quality to the Tacotron. Therefore, our library was developed in such a way to allow switching between Tacotron and Deep Voice 3, while using the same vocoder.

3) Griffin-Lim Algorithm: Finally, recovery of the audio signal from the spectrogram is done using the Griffin-Lim Algorithm (GLA; [14]), based on the short-time Fourier transform (STFT) redundancy. The resulting spectrograms are consistent, due to the STFT's redundancy. The GLA is focused on consistency and does not take into consideration any prior knowledge of the target signal.

C. Architecture of the library

Since the aim of our work was to design a library, for the social robot platform, a simple API was needed to configure the model, send text inputs to run on the model, and a response mechanism to return the generated speech data, or play it. To maximise modularity, the code was split into *synthesizer*, *text processor*, *audio processor* and *request handler* components.

1) Text Processor: The text processor takes the text (input) from the request handler and converts it to a vector of numeric values, to be fed to the network. Raw text input is preprocessed. First, the text is converted to ASCII form, then punctuation is added to the end of the sentence (if not present). Text is converted to lower-case letters. Common abbreviations are expanded, while numbers and currency symbols are turned into text, for the synthesizer to be able to say them. Finally, unwanted white-spaces are trimmed. This text is then converted to numbers. However, we added also possibility of using ARPAbet [10], which provides a phonetic form of a word, or a sentence, to be pronounced in an expected way.

2) Synthesizer: Synthesizer runs the neural network model and synthesizes the audio waveform, from the input text. This part of the library contains configurable parameters, which are, initially, set up to balance speech quality and inference time. The inferencer can run on the Tensorflow, or the PyTorch, frameworks. Once the synthesizer is loaded, an initial run of the model might be required to "warm it up". This can lead to longer run time of the first run.

3) Audio Processor: The audio processors' role is to apply low pass filters, or audio gain adjustment, to the raw waveform generated by the synthesizer, and to assure that the audio playback is carried smoothly. It also provides direct control for the request handler to interrupt, or queue, multiple audio fragments to be played.

4) Request Handler: This component is responsible for handling HTTP, or Message Queuing Telemetry Transport (MQTT), requests. Note that the MQTT protocol is lightweight and thus more suitable for the Internet of Things messaging. The created application stack uses an MQTT broker running in the background, allowing subscription for the "TTS" topic requests, from any other application.

D. Inference pipeline

The inference pipeline is embedded within the edge device that is the core of the social robot. It is to communicate with users through voice commands, and expected to "talk back". When the robot shall say a text, it is generating a TTS command for the TTS pipeline, to play audio as soon as possible.

The TTS pipeline has two goals: a) to perform TTS processing (preprocessing input text, encoding it, generating spectrograms, and audio waveforms) on the device itself, without use of external resources, and b) to perform the TTS task in *real-time*, i.e. the total time of processing the pipeline for a single sentence should not exceed the total time of generating waveform for that sentence. The delay between initiating TTS processing, and start of hearing voice was imposed to be 1 second (or less). This value corresponds to known findings in Human-Robot Interaction studies [19].

IV. Implementation of model

Let us now present the details of the implemented model, and the optimizations performed to attain low latency speech synthesis. This optimization is achieved by speeding-up matrix operations of the network, and using third-party libraries to speed-up math operations in Python, to achieve speeds similar to those obtainable in C++.

For their speed and quality, Google's Tacotron and Baidu's Deep Voice 3 were selected, for the developed library. All optimizations were done post-training, to the frozen models, using techniques specified below.

A. Synthesizers

The Tacotron architecture was used for the primary default synthesizer, with few changes made to suit our needs. The fundamental structure of Tacotron is a seq2seq model with attention [20], [21]. It takes characters as input, and generates spectrogram frames, used to create a waveform data using a vocoder. Unlike the original model, we use the LSTM cells in the attention and decoder units, to compensate for the memory loss that will occur when the model is optimized. Figure 1 shows the components that make up this model.

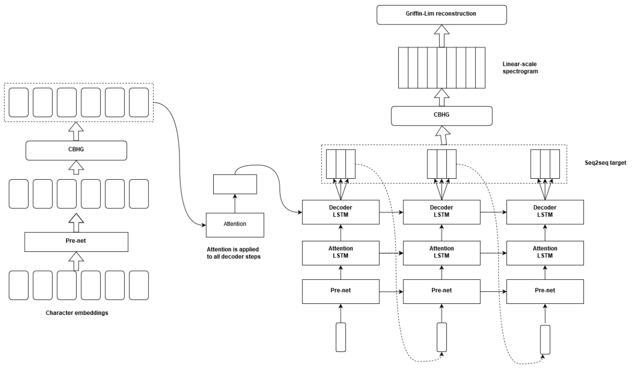

Fig. 1. Tacotron based architecture with bi-directional LSTMs

The second synthesizer is the Deep Voice 3 architecture, which consists of three parts that are similar to the Tacotron, but differ in the type of network used. Deep Voice 3 is a fully-convolutional model. To ensure minimal attention errors, in the production TTS system, a monotonic attention mechanism, as described in [22], instead of the usual attention mechanism, has been applied. The structure of this model is presented in Figure 2.

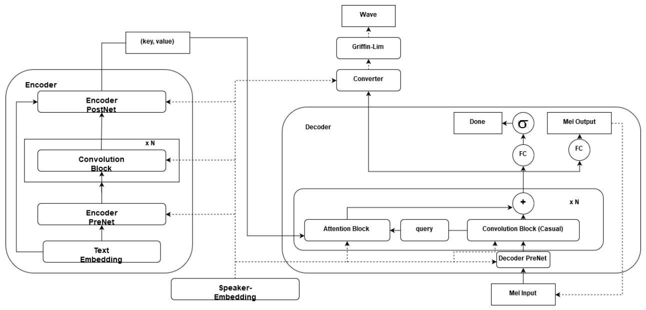

Fig. 2. Architecture of the Deep Voice 3 based model

B. Vocoder

The phase signal reconstruction, from the spectrogram output of the decoder, is done using the GLA. When given a magnitude spectrogram, to produce time-series data, the algorithm works in the following manner.

1) Initialize complex matrix with magnitude values as real part, and uniform noise values as imaginary part. This gives amplitude value, but no phase information.
2) Apply inverse Short-Time Fourier Transform (ISTFT) to get time series data from amplitude information.
3) Apply Short-Time Fourier Transform (STFT) to the time series, to obtain initial phase information, which will be inaccurate as the time series is of low quality.

4) Replace the real values received from the STFT with the original magnitude values to preserve the amplitude.
5) Iterate over steps 2-4, each time gaining more phase information.
6) Stop when the phase information is satisfying, or when the process converges.

What needs to be assured is that the duration of the output signal does not differ a lot from the original time series. Since above steps involve multiple matrix operations, they are time consuming. Hence, Tensorflow implementation of STFT and ISTFT that utilizes GPU (if available), and CPU-optimized implementations of STFT and ISTFT, were applied.

V. Data used in experiments

Like for any neural networks, the quality of results depends on the quality of the data used to train it. Hence, we had to provide several hours of audio data, along with their text transcriptions. We had to ensure that the training data has a diverse combination of words, to assure that the model "can handle" the text on "any" topic. For initial evaluation, the model was trained on a single speaker data. Obviously, multi-speaker scenario is much more complex, and is outside of scope of current contribution.

Usually, for speech synthesis, the English language is used and training involves the, so called, Linda Johnson (LJ) Speech dataset. This is a public dataset, consisting of 13,100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription is provided for each clip. Clips vary in length from 1 to 10 seconds, and have a total length of approximately 24 hours. The texts were published between 1884 and 1964, and are in the public domain. The audio of the texts, read by Linda Johnson, was recorded in 2016-17.

Although the files are good quality, their content is outdated, and does not contain words that are commonly used today. This is unacceptable for modern social robots. To overcome this issue we needed a dataset with a wider vocabulary. Since we already had 24 hours of audio data we decided to add several more hours of new data. This data was obtained from BBC, in the form of articles from news website corresponding to stories, in five topical areas, from 2004-2005. The areas are business, entertainment, politics, sports, technology. This dataset consisted of 2225 documents, with each containing few short articles. Before we could train the model the data needs to be split into the form of sentences so each sentence and it's audio can be used as a [text,audio] sample pair provided in a comma separated file for the training code. Since it is not possible to generate the BBC text dataset's audio using the same voice as LJ dataset, we had to generate all sentences from *both* datasets using the same voice, to ensure consistency. This was achieved by extracting the text data from the LJ dataset and combining is with the text from the BBC dataset.

Also, a preliminary analysis was done for the dataset, to understand the lexical diversity. Lexical diversity is a ratio between the number of unique words (excluding stop words such as "the","a", etc., and non-alphabetic characters) and the total number of words. This was done using the NLTK (Natural Language Toolkit) library.

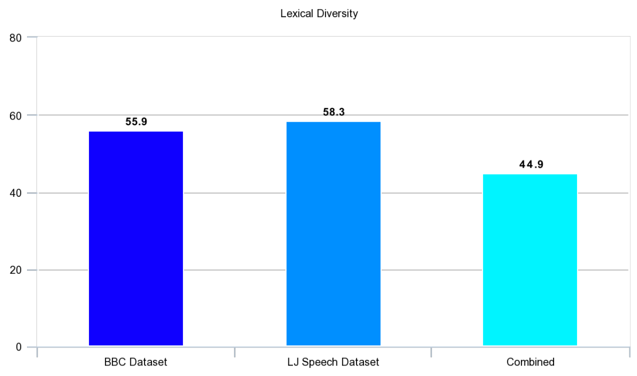

Fig. 3. Lexical diversity comparison for speech datasets

TABLE I
Comparison of the speech training datasets

Feature	**Datasets**	
	LJSpeech	***BBC***
Duration	23.4 hours	26.7 hours
Unique Words	14986	17058

Figure 3 shows the lexical diversity of the two datasets (LJ and BBC) individually and after combining. The notable drop in lexical diversity for the combined dataset is due to the sheer volume of the text. Table I further illustrates the statistics of the text dataset and the duration of generated audio files, which were used for training. Slightly more audio of the BBC dataset was used, as it contained a more diverse set of words that are suitable for a modern vocabulary.

For the combined dataset, we applied basic preprocessing. Using the NLTK, articles were tokenized to sentences. Unwanted expressions, and symbolic characters were removed. Sentence-tokens that were too small to be spoken sensibly, or that were non-unique (like 37m/h – meaning meters or miles per hour) were removed.

To provide a voice for the collected data, the Google's Cloud Text To Speech engine was used, as it provides high quality natural sounding Wavenet voices. It uses large Text-To-Speech models, on GPU farms, running Wavenet vocoders in real time. Note that this makes then useless for our assumed scenario. All audio was turned into WAV files with constant 32kbps bit rate, and sampling rate of 24,000Hz, varying from 1 to 15 seconds in duration each. The complete dataset contained 24,837 audio files with a total size of 8.3 GB and total duration of 50.1 hours.

VI. Model training and optimization

The training was carried out on Microsoft Azure Cloud. The audio files were preprocessed to generate numpy array files, with spectrogram information, which was used for training Tacotron, and Deep Voice 3 models, as ground truth for the text input provided for the corresponding audio files. Figure 4 shows what is expected from the different stages of training process of an encoder-decoder model. The encoder and decoder time steps should match, forming a clear diagonal line. This denotes that there is a direct correlation between the input and the output, at the same time step. The speech was "sounding human" in around 3000 steps, and started being intelligible after approximately 6000 steps. It was trained for a total of 30000 steps, in the final model, used for live tests.

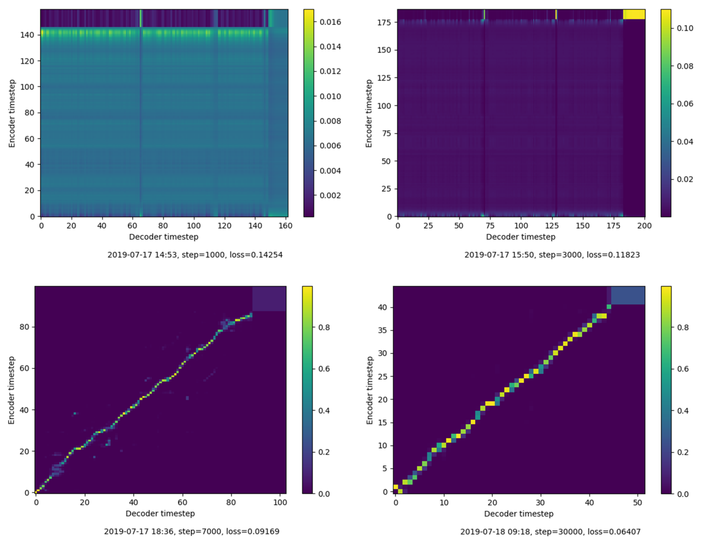

Fig. 4. Encoder and decoder alignment progress in training

The training lasted 8-10 hours, for the hyperparameters summarized in Table II. Note that a lower batch size, or learning rate, like 16 and 0.001, respectively, can make the training process take over 10 hours to complete. Increasing the batch size allows the model to take more samples at a time and allows it to learn faster. Increasing the sampling rate also makes the training slower but can give better quality sound. Increasing the batch size and learning rate over the above given values resulted in model "not learning". The time taken for each step, for the Deep Voice 3 model, was considerably lower than the Tacotron, so the step count is not a representative parameter. The reported Adam optimizer beta 1 and 2 values provided the fastest results with reasonable accuracy.

TABLE II
TRAINING PARAMETERS FOR SYNTHESIZER MODELS

Parameter	Tacotron	Deep Voice 3
Mel frequency banks	80	80
Sampling Window	1024	1024
Batch Size	32	16
Sampling Rate	24000	22050
Initial Learning Rate	0.002	0.0005
Beta 1	0.9	0.5
Beta 2	0.999	0.9
Training Iterations	33000	640000

A. *Optimization of trained model*

There were several methods applied to optimize the model. Let us now describe them in some detail.

1) Freezing and Pruning: Freezing the graph is the most basic form of optimization. The model files have the weights represented as constant operations in the graph definition, and as Variables in the checkpoint files. Redundant storage can be removed by combining the two, reducing the model size. This eliminates the requirement that weights be loaded and combined separately, resulting in a lower overhead while loading the model. This procedure decreases the model's total size by a reasonable amount. After freezing, transforms can be applied to the graph.

(A) *Node removal*. Here, single-input-single-output nodes can be removed and their actions delegated to the preceding nodes. This operation is "dangerous" because it's possible that removing some nodes may change the output of the model, so model accuracy check is required.

(B) *Merging duplicates*. Specifically, constant nodes with the same types and contents, or nodes with the same inputs and attributes, can be merged. This operation can be useful in a graph with a lot of redundancy.

(C) *Stripping unused nodes*. This removes all nodes not used in calculating the output layers fed by inputs. For instance, this removes training-only nodes, like save-and-restore, or summary, operations.

(D) *Folding*. Fold constant nodes operation, searches for any sub-graphs, within the model, that always evaluate to constant expressions, and replaces them with those constants. This optimization is always executed at run-time, after the graph is loaded. Hence, running it "offline" won't help latency, but it can simplify the graph. Folding of batch normalization nodes scans the graph for any channel-wise multiplies and applies them so that this can be omitted at the inference time. This is performed after folding the constants, as the pattern can be spotted only if the normal complex expression collapsed down into a simple constant.

2) Third party libraries: During the training, a Google-developed TCMalloc library [23] was used to speed up the memory allocation, thread caching, and garbage collection. This allowed for each training step to be about 40% faster. Usually each thread has its own arena of space allocated. When it finishes, the next thread cannot use the space and allocates the needed space elsewhere. TCMalloc allocates a thread-local cache instead. As needed, objects are moved from the central data structures to the thread-local cache, and periodic garbage collections migrate memory back.

A GPU supported implementation of GLA was used within the Nvidia Jetson platform. Moreover, LibROSA, package for audio analysis was used to implement fast CPU version of the Griffin-Lim algorithm, for the Raspberry Pi. To handle matrix operations, during runtime, Numba Python compiler, with the LLVM library was used.

VII. Evaluation of results

The research objective was real-time speech synthesis, on a single board computer. It was expected that, for example, an audio of 5 seconds will be synthesized, on the device, within 5 seconds or less. This goal was, mostly, achieved.

A. Speech synthesis quality and speed

Realistically speaking, speech *quality* needs to be evaluated by humans, to judge factors like intelligibility, prosody, intonation and pronunciation. To remain objective, we focus on spectrogram analysis, to explain changes in quality, as related to the vocoder parameter. Here, the parameter is the number of Griffin Lim iterations, used while recreating the audio waveform from the network generated spectrogram. This is the main factor (other than model optimization) that defines the balance between speed and quality.

In Figure 5 we present the spectrograms of the same sentence being spoken by three different models. The top is the Tacotron with 5 Griffin Lim iterations, the second is also the Tacotron but with 60 Griffin Lim iterations, and the last one is the Deep Voice 3 with 60 Griffin Lim iterations.

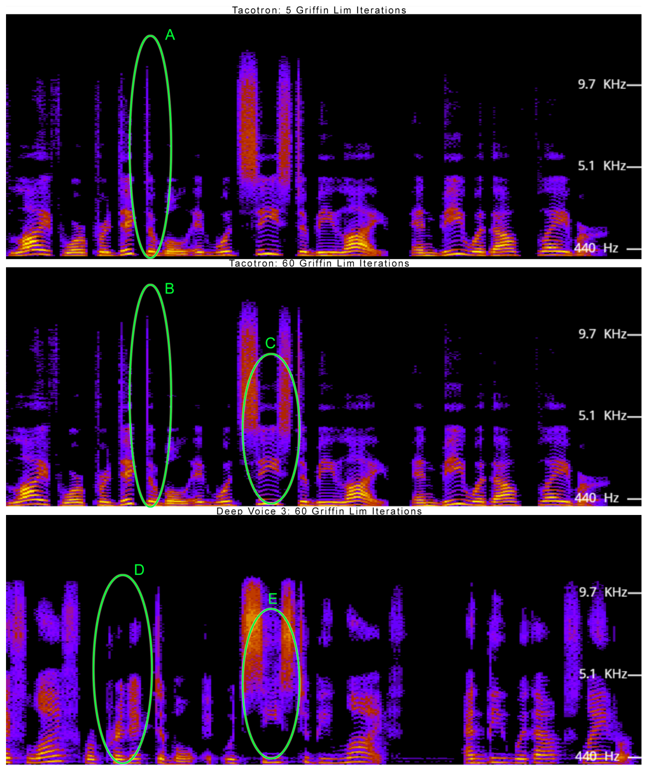

Fig. 5. Spectrogram comparisons for synthesizers

In areas marked A and B, there is a noticeable clarity in the onset of the frequency spectrum, as B compared to A. This is heard even in the audio, as the words are clearly separated and have a distinct onset. Comparing areas marked B and C, with D and E, we see a lot of missing frequencies in area D, and a mixture of several frequencies in area E. This is also visible in other areas that are not marked. When we compared the generated audio, this difference was pretty evident, as there was no clear distinction between different sounds.

As illustrated in Figure 6, time taken for the model to synthesize the audio of length 6.8 seconds, on different devices, varying the number of Griffin Lim iterations, differs mostly in CPU only tests, when compared to the devices with the GPU. The performance on the CPU, without any acceleration, dropped exponentially, as the number of iterations increased. The Jetson Nano, on the other hand, is able to deliver real-time performance, even when the iteration number increased. This is, in part, due to the fact that the Griffin Lim algorithm uses Tensorflow and can run in parallel on the GPU.

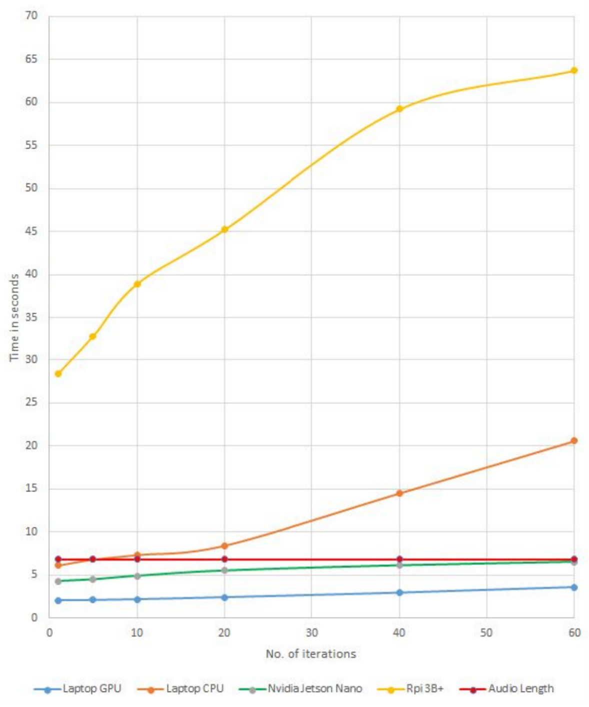

Fig. 6. Inference speed comparison chart

B. Optimization results

With the optimizations summarized in Section VI-A1 the size and complexity of the model was reduced by almost 3 times, as summarized in Table III. Hence, the model loads faster and is portable across resource constrained devices, while the quality remains indistinguishable to the human ear.

TABLE III
Model optimization stats

Features	**Optimization**		**GL Iterations**	
	Before	***After***	***20***	***60***
Network Nodes	12295	1069	3400	7851
Model File Size	89.31 MB	30.02 MB	30.53 MB	31.51 MB

VIII. Concluding remarks – limitations

This contribution summarizes initial work on delivering Speech-to-Text capabilities to social robots, using single board computer systems with limited resources. By combining, applying and optimizing existing algorithms/techniques promising results have been achieved.

However, we have noted a number of limitations of the developed solutions (even in their most advanced versions). (1) The trained model lacks speed when running on pure CPU based machines. This is caused, primarily, by the requirements of the Griffin-Lim algorithm. (2) While the quality of speech is great, for a portable real-time model, it does not have capabilities of the newer generation vocoder solutions [24]. These vocoders produce a lot of acoustic features of human speech that algorithms, considered in this work, cannot compete with. (3) There is less customizability with the speaker voice, as only single speaker TTS models were tried. (4) The sensitivity of the model does not allow for lowering the floating-point precision of the weights (post-training). This further limits its speed on CPU architectures. (5) The model lacks ability to specify the stress for each syllable of the input sentence, based on emotion of the context. Since the English language rules are not very clear when we consider phonetic consistency, there is no easy way to, for instance, add a rule-based preprocessing for automating this task. (6) The initial overhead for loading the model can cause delays, if the TTS module is frequently enabled and disabled to free up the processing power for other tasks on the platform. This can be the case when system resources need to be allocated for computer vision tasks. Delay time can go up to 30 seconds when the TTS module is deallocated and brought back. The same is the case if the users want to dynamically switch, or change, the parameters of the model that they wish to use.

These limitations can be seen also as a research program that we plan to delve into to deliver the needed (by the SmartLife company) solution. We will report on our progress in subsequent publications.

Acknowledgment

This work is a result of direct collaboration between the SmartLife Systems Sp. z.O.O. and the Faculty of Mathematics and Information Sciences of the Warsaw University of Technology. All the resources required for the development, training and testing of the models were provided by SmartLife Systems Sp. z.O.O.

References

[1] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, and R. Pang, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[2] A. W. Black and K. A. Lenzo, "Flite: a small fast run-time synthesis engine," in *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.

[3] Z.-H. Ling and R.-H. Wang, "HMM-based unit selection using frame sized speech segments," in *Ninth international conference on spoken language processing*, 2006.

[4] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura, "Speech parameter generation algorithms for HMM-based speech synthesis," in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, vol. 3. IEEE, 2000, pp. 1315–1318.

[5] W. Hamza, E. Eide, R. Bakis, M. Picheny, and J. Pitrelli, "The IBM expressive speech synthesis system," in *Eighth International Conference on Spoken Language Processing*, 2004.

[6] M. Charfuelan and I. Steiner, "Expressive speech synthesis in MARY TTS using audiobook data and emotionML." in *INTERSPEECH*, 2013, pp. 1564–1568.

[7] M. Schröder, P. Baggia, F. Burkhardt, C. Pelachaud, C. Peter, and E. Zovato, "EmotionML–an upcoming standard for representing emotions and related states," in *International Conference on Affective Computing and Intelligent Interaction*. Springer, 2011, pp. 316–325.

[8] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[9] B. Potard, M. P. Aylett, D. A. Baude, and P. Motlicek, "Idlak Tangle: An Open Source Kaldi Based Parametric Speech Synthesiser Based on DNN." in *INTERSPEECH*, 2016, pp. 2293–2297.

[10] S. Imai, "Cepstral analysis synthesis on the mel frequency scale," in *ICASSP'83. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 8. IEEE, 1983, pp. 93–96.

[11] B. Series, "Method for the subjective assessment of intermediate quality level of audio systems," *International Telecommunication Union Radiocommunication Assembly*, 2014.

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

[13] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, and S. Bengio, "Tacotron: Towards end-to-end speech synthesis," *arXiv preprint arXiv:1703.10135*, 2017.

[14] N. Perraudin, P. Balazs, and P. L. Søndergaard, "A fast Griffin-Lim algorithm," in *2013 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE, 2013, pp. 1–4.

[15] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv:1609.03499 [cs]*, Sep. 2016, arXiv: 1609.03499. [Online]. Available: http://arxiv.org/abs/1609.03499

[16] Y. Wang, D. Stanton, Y. Zhang, R. J. Skerry-Ryan, E. Battenberg, J. Shor, Y. Xiao, F. Ren, Y. Jia, and R. A. Saurous, "Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis," *arXiv preprint arXiv:1803.09017*, 2018.

[17] R. J. Skerry-Ryan, E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark, and R. A. Saurous, "Towards end-to-end prosody transfer for expressive speech synthesis with tacotron," *arXiv preprint arXiv:1803.09047*, 2018.

[18] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep voice 3: Scaling text-to-speech with convolutional sequence learning," *arXiv preprint arXiv:1710.07654*, 2017.

[19] T. Shiwa, T. Kanda, M. Imai, H. Ishiguro, and N. Hagita, "How Quickly Should a Communication Robot Respond? Delaying Strategies and Habituation Effects," *International Journal of Social Robotics*, vol. 1, no. 2, pp. 141–155, Apr. 2009. [Online]. Available: https://doi.org/10.1007/s12369-009-0012-8

[20] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[21] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *arXiv preprint arXiv:1511.06391*, 2015.

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

[23] S. Ghemawat and P. Menage, "TCMalloc : Thread-Caching Malloc," 2019, publisher: Google. [Online]. Available: http://goog-perftools.sourceforge.net/doc/tcmalloc.html

[24] G. Yang, S. Yang, K. Liu, P. Fang, W. Chen, and L. Xie, "Multi-band MelGAN: Faster Waveform Generation for High-Quality Text-to-Speech," *arXiv preprint arXiv:2005.05106*, 2020.