10th ISCA Speech Synthesis Workshop
20-22 September 2019, Vienna, Austria

Building Multilingual End-to-End Speech Synthesisers for Indian Languages

Anusha Prakash[‡][*], *Anju Leela Thomas*[§][*], *S. Umesh*[‡], *Hema A. Murthy*[§]

[‡]Dept Of Electrical Engineering, Indian Institute of Technology Madras, Chennai, India
[§]Dept Of Computer Science & Engineering, Indian Institute of Technology Madras, Chennai, India

anushaprakash90@gmail.com, anjuthomas95@gmail.com, umeshs@ee.iitm.ac.in, hema@iitm.ac.in

Abstract

Building text-to-speech (TTS) synthesisers is a difficult task, especially for low resource languages. Language-specific modules need to be developed for system building. End-to-end speech synthesis has become a popular paradigm as a TTS can be trained using only <text, audio> pairs. However, end-to-end speech synthesis is not scalable in a multilanguage scenario, as the vocabulary increases with the number of different scripts.

In this paper, TTSes are trained for Indian languages using two text representations– character-based and phone-based. For the character-based approach, a multi-language character map (MLCM) is proposed to easily train Indic speech synthesisers. The phone-based approach uses the common label set (CLS) representation for Indian languages. Both approaches leverage the similarities that exist among the languages. The advantage is a compact representation across multiple languages. Experiments are conducted by building TTSes using monolingual data and by pooling data across two languages. The ability to synthesise code-mixed text using the phone-based approach is also assessed. Subjective evaluations indicate that reasonably good quality Indic TTSes can be developed using both approaches. This emphasises the need to incorporate multilingual text processing in the end-to-end framework.

Index Terms: end-to-end speech synthesis, Indian languages, multi-language character map, multilingual synthesis, code-mixing

1. Introduction

Developing a good quality text-to-speech (TTS) synthesiser is a challenging task, especially when dealing with multiple languages in a low-resource scenario. India has 1652 languages[1] with 22 official languages written in different scripts. Due to the lack of adequate amount of data and linguistic tools, Indian languages are considered to be low resource. Building a conventional TTS system requires a certain amount of feature engineering, a parser for parsing text into sub-word units (phones, syllables), an alignment module, etc. With the success of attention-based sequence to sequence (seq2seq) models in an end-to-end framework, building a TTS system is simple [1, 2, 3].

TTS systems have been built in a cross-lingual manner in our earlier work [4]. In [4], code-switchable TTSes are built using a mixture of data from two languages. Initial experiments were conducted in the current work by similarly pooling data across two languages. Tacotron2 [2] was used as the end-to-end technique. Systems built using data of one language gave reasonably good quality synthesised speech. But when data of two languages were combined (where the graphemes corresponding to both languages occupy different UTF-8 representations), the quality of synthesised speech was poor[2]. This indicates that a purely end-to-end approach is not adequate and additional processing is required.

There are two issues that we address in this paper, namely, the issue of different grapheme representations for Indian languages, and the issue of low digital resources, in that the amount of data available for each language is inadequate for building end-to-end systems. The common phonetic characteristics of different languages enable the use of pooled resources across languages, where pooling is performed based on the phonetic content. Motivated by the work in [5, 6, 7], where similarities among Indian languages are studied and exploited, the focus is on capitalising on these similarities for system building. In this context, two types of text representations are explored in the end-to-end framework– character-based and phone-based. The goal is to enable effortless training of Indic systems for monolingual and multiple language data.

The pipeline in Tacotron2 [2] takes character sequences as input and generates the speech waveforms. When dealing with multiple languages, it may be required to process linguistic representations across languages. In the literature, efforts in training TTSes in the seq2seq framework have explored various linguistic representations. In [1] and [2], the input text is processed in terms of its constituent characters. [8] explores convolution attention based models for multilingual speech synthesis of Indian languages and studies how the multilingual system scales for new languages and new speakers. In [8], the UniTran SAMPA table is used to map Unicode characters to the SAMPA symbol-set. [9] uses a neural multi-speaker TTS system for conversions between English, Spanish, and German. It uses a per-language encoder to process input sequences in terms of phonemes. [10] uses a phoneme-based representation with vowels using three different symbols depending on their level of stress.

Some implementations have also explored representation mixing [11, 12]. [11] processes the input text in terms of characters, phonemes and stresses and also explores a joined representation of characters and phonemes. [12] enables a choice between character, phoneme, or mixed representations during inference. In [13], a multilingual speech synthesis system is trained using byte representations across English, Mandarin and Spanish. The text is processed using Unicode bytes, especially, the UTF-8 variable length byte sequence for each character. This provides a very compact representation as the vocabulary size of Unicode bytes is always 256.

In [14], code-switchable TTSes are built using a mix of monolingual Mandarin and English data by encoding the input text in two ways: (1) using a separate encoder to encode each language, and (2) using a shared encoder with language

[*]Both authors have contributed equally to this paper.
[1]This statistics is from the 1961 Census report of India.
[2]Synthesised examples can be found at this link: https://www.iitm.ac.in/donlab/ssw2019/mlcm_phone/index.html

10.21437/SSW.2019-35

embedding.

In the current work, we explore character-based and phone-based representations in the default Tacotron2 architecture [2]. For Indian languages, the text is in UTF-8 representation. In Tacotron2, the text in UTF-8 format is first processed into individual characters based on Unicode values. This results in a character map that maps all the distinct characters in the given text to a set of tokens used for further processing. Each character in each script has a distinct Unicode value. When building a TTS synthesiser using multiple languages, the total number of unique characters increases. The Unicode byte representation also does not account for similarities across Indian languages. Therefore, a very simple and compact character representation for Indian languages is proposed. The multi-language character map (MLCM) is a superset of characters across 13 Indian languages spanning 8 different written scripts wherein characters that sound similar across languages are given the same representation. In the current work, the character map is replaced by a pre-defined MLCM. TTSes are trained using Tacotron2 in an end-to-end framework and their performance is compared with those trained using language-specific character maps. The effectiveness of systems trained using MLCM is evaluated on (1) monolingual data (2) mixture of data from two languages.

To obtain the phone-based representation from the input text, the unified parser for Indian languages is used [15]. Phones are represented in terms of common label set (CLS) notations [6]. The performance of phone-based systems is assessed only on monolingual data. Experiments are performed with two Indo-Aryan (Bengali, Hindi) and two Dravidian (Malayalam, Tamil) languages. The performance of both types of text representations is also compared. This kind of text processing results in a compact representation.

An interesting application of the phone-based experiments is code-mixing. Indians are inherently multilingual and tend to mix different languages in a conversation. One of the most common languages that is mixed with the native tongue is English. A code-mixable TTS is developed in the current work using a mixture of two monolingual datasets. This is similar to the approach in [4]. Code-mixing is the alternation between languages in a single sentence[3]. A *Hindi+English* TTS is trained using monolingual Hindi and English data. As English has a different set of characters from Hindi, English is also represented in terms of CLS labels using the technique in [4]. Additionally, a classification and regression tree (CART) is built so that out-of-vocabulary (OOV) words are also handled. A phone-based *Hindi+English* TTS is trained, and its performance is compared with monolingual Hindi and English TTSes by evaluating on a set of code-mixed Hindi and English text.

The rest of the paper is organised as follows. Both text representations are described in Section 2. The proposed multi-language character map is presented here. Section 3 describes the end-to-end framework for training a TTS. Section 4 details the experiments carried out and the results obtained. The work is concluded in Section 5 with directions for future work.

2. Text representations

2.1. Character representation: multi-language character map (MLCM)

The input text is in UTF-8 format as shown in the second column of Table 2. Tacotron2 generates a character map wherein distinct characters (extracted from the input text) are mapped to a set of tokens needed for further processing. It includes representations for unknown entities (<unk>), punctuation marks and language-specific characters. Out-of-vocabulary characters that appear during synthesis are considered to be unknown entities. Spaces and punctuation marks, including periods, commas, question marks are considered as separate characters.

Unlike English which has 26 distinct characters, scripts in Indian languages have typically around 55-70 characters based on their Unicode values. Features of Indic scripts share many similarities. Characters in a script are arranged in a specific structure. Vowels come first followed by consonants arranged based on their place and manner of articulation. These are followed by semi-vowels and fricatives. There is another class of characters referred to as vowel modifiers, that when preceded by consonants generate different combinations of graphemes. Exploiting these similar features, a superset of character maps has been developed.

Table 1: *Examples of cross-lingual mapping in MLCM. Pronunciations are in terms of CLS labels.*

Pronuncation in CLS	Bengali	Devanagari	Gujarati	Kannada	Malayalam	Odiya	Tamil	Telugu
a	অ	अ	અ	ಅ	അ	ଅ	அ	అ
aa	আ	आ	આ	ಆ	ആ	ଆ	ஆ	ఆ
	◌া	◌ा	◌ા	◌ಾ	◌ാ	◌ା	◌ா	◌ా
i	ই	इ	ઇ	ಇ	ഇ	ଇ	இ	ఇ
	◌ি	◌ि	◌િ	◌ಿ	◌ി	◌ି	◌ி	◌ి
ka	ক	क	ક	ಕ	ക	କ	க	క
ga	গ	ग	ગ	ಗ	ഗ	ଗ	-	గ
zha	-	-	-	-	ഴ	-	ழ	-

The multi-language character map is a manually prepared character map for 13 Indian languages with 8 distinct scripts (Bengali, Devanagari, Gujarati, Kannada, Malayalam, Odia, Tamil, Telugu). Similar characters across these scripts are given the same representation. This is similar to Bharati script which is a common script representing characters in nine major Indian scripts [16]. This script has been used to develop a multilingual optical character recognition system for Indian languages [17]. But the Bharati script introduces a new set of characters which needs to be learnt. Vowel modifiers do not have separate representations in the Bharati script, while this distinction is made in the Unicode representation.

In MLCM, vowels and vowel modifiers of the same script are mapped together as they represent the same characters. The character to token mapping is many-to-one. This mapping has been prepared based on the structure of the scripts and the common label set (CLS) representation for Indian languages [6]. This is a compact representation across languages as the size of the token set is just 68 including <unk>, full stop, comma, blank and 64 character tokens. Other punctuation marks are not included, as Indian language texts are rarely punctuated. This kind of cross-lingual mapping is especially useful when dealing with data pooled across languages with different scripts. MLCM can be easily extended to include characters of a new Indian language.

Examples of mappings in MLCM are given in Table 1. Pronunciations of the characters are in terms of CLS labels. As seen from the table, "aa" and "i" have two rows each– the first one for corresponding vowels and the second for vowel modifiers. A hyphen in the figure denotes that the corresponding character is not present in that script. In the language-specific character

[3]Code-mixing is an intra-sentential phenomenon whereas code-switching is an inter-sentential phenomenon

map, characters in different scripts are considered as distinct characters and processed as such. In MLCM, characters across scripts are mapped together, as illustrated in Figure 1.

2.2. Phone representation: common label set (CLS)

In the phone-based approach, the input text is first parsed into constituent phones, which are denoted by labels in the common label set (CLS) [6]. CLS provides a standard notation of phones across different Indian languages in Latin-1 script. A unified parser is used to convert words in Indian languages to CLS representation [15].

The UTF-8 to Latin-1 mapping is not one-to-one in CLS. Hence, two approaches are explored in the context of phone-based representation– transliteration and phone mapping. In the former, the unified parser is used to transliterate words into CLS notations. A phone may be represented by more than one Latin-1 character. For example, long "a" is denoted by "aa". This transliterated text is then used directly in the subsequent stages. Tacotron2's character map, in this case, consists of Latin-1 characters present in the training data mapped to a set of tokens.

Table 2: *Examples of phone-based representations– transliteration and phone mapping*

Language	Word	Transliteration	Phone mapping
Bengali	শ্রীরামকৃষ্ণ	shriiraamkrqsxnx	शrIrAmkRषण
Hindi	अध्येता	adhyeetaa	aधyEtA
Malayalam	പ്രതിരോധത്തിന്	pratiroodhattin	pratirOधattin

In the second approach, individual phones from the unified parser's output are mapped to a single unique character. This is to ensure that system building is purely phonetic. For example, long "a" is mapped to "A" which is a single character. The resultant character may be in Latin-1 or UTF-8 format. The modified input text is presented to Tacotron2, and the character map has a combination of Latin-1 and UTF-8 characters. Examples of words parsed in these two formats are given in Table 2.

The cardinality of CLS is 83. Individual Indian languages have around 49-58 phones, except for Tamil which has around 35 phones. The cardinality of the phone-based system is equal to the number of phones covered by the data. On the other hand, the cardinality of the transliteration system is around 22-25.

For code-mixing, English words need to be handled differently. There is no one-to-one mapping of English characters to characters in Indian scripts. Hence, the MLCM approach is not feasible for English text. Similar to the technique in [4], English words are parsed into CLS notations using a classification and regression tree (CART). The English text is then modified using the phone mapping approach.

3. Training the end-to-end TTS synthesiser

ESPNet's implementation [18] of Tacotron2 is used in this work[4]. The basic architecture of Tacotron2 is an attention-based encoder-decoder network with a WaveNet vocoder [2]. Only <text, audio> pairs are required for training the TTS. The text is first processed into a sequence of characters (or phones) and mapped to tokens. If the text representation is character-based, then this mapping is done using MLCM. Each character (or phone) is then represented by a one-hot vector and embedded into a continuous vector. This sequence of embedded vectors is fed as input to the encoder, which produces sequential representations of the text.

The encoded sequence is fed to the attention module which summarises it as a fixed-length context vector. This is required by the decoder for each step. In [2], location-sensitive attention is used. In the current work, forward attention with a transition agent is used [19]. For each decoder step, only monotonic alignments are considered, as the nature of alignment between linguistic and acoustic sequences is monotonic. The decision to stay or proceed at each decoder step is based on the transition agent. The decoder predicts mel spectrogram frames at each step based on the encoded input. The encoder-decoder mechanism is trained thus from the speech data.

During synthesis, the input text is processed in terms of individual characters (or phones), embedded and then passed through the encoder-decoder network. In Tacotron2 [2], a WaveNet vocoder auto regressively predicts the speech waveform sample-wise by conditioning on the generated mel spectrograms. As a robust WaveNet vocoder requires at least tens of hours of data for training, the Griffin-Lim algorithm is used in this work [20]. The Griffin-Lim technique also allows for faster experimentation. The generated mel-scale spectrogram is first converted to a linear-scale spectrogram. The Griffin-Lim algorithm reconstructs the speech waveform from the linear-scale spectrogram by iteratively estimating the phase. This is similar to the implementation in vanilla Tacotron [1].

4. Experiments and results

Experiments are carried out with female datasets from Indic TTS database [21] for four Indian languages, namely, Bengali, Hindi, Malayalam and Tamil. Around 5 hours of data is chosen for each language. The text is in UTF-8 format. Details of the data are given in Table 3. Each monolingual dataset is the speech data of a single speaker. The Tacotron network does not receive any speaker or language identity as input.

Table 3: *Details of the Indic datasets*

Dataset/ Language	Language family	Script	Number of utterances
Bengali	Indo-Aryan	Bengali	3179
Hindi	Indo-Aryan	Devanagari	2178
Malayalam	Dravidian	Malayalam	3418
Tamil	Dravidian	Tamil	1585

Before training, the text is cleaned up. All intermediate punctuation marks except comma are removed. The end of every sentence is marked with a period.

Preliminary experiments were carried out to train reasonably good TTS systems for the four datasets. It was observed that training was very sensitive to the input data. For example, a few misplaced commas in the training text were enough to degrade the quality of synthesised speech. Further, in batch mode, sorting based on the length of the input character sequence resulted in better synthesis quality, compared to shuffling the data randomly. For each system, 90% of the data is considered as train data and the rest as validation data.

Subjective evaluations are conducted to assess the performance of TTSes. The first set of experiments evaluate character-based systems, specifically analysing the effectiveness of using MLCM. The next set of experiments compare the two types of phone representations followed by the comparison

[4]Link to ESPNet's code: `https://github.com/espnet/espnet`

between character and phone-based systems. The phone-based approach is also evaluated on code-mixed sentences.

Out-domain sentences are used for evaluation. They include test sentences from Blizzard Challenges 2013-15 [22, 23, 24] and sentences taken from the web. 10 native listeners of the language are used in each test. Evaluators are in the age group 20-35 and have no known hearing defects.

4.1. Character-based experiments

4.1.1. Mapping together vowel modifiers and vowels

The default Tacotron implementation generates a character-to-token mapping based on Unicode values. It does not take into consideration the acoustic similarity of characters. Vowel modifiers are characters that are appended to consonants to generate different graphemes. For example, vowel modifier *aa* appended to character *k* results in the sound "kaa". This is the same as appending character *k* and vowel *aa*. This distinction is made only during writing. Hence, ideally, a vowel and the corresponding vowel modifier should be mapped to a single token, although they have different Unicode values.

Two Malayalam TTS systems are built with 5 hours of data using– (1) the default character map generated by Tacotron, in which vowels and vowel modifiers are represented separately, and (2) a modified character map in which the vowel modifiers and the corresponding vowels are mapped to the same token. A pairwise-comparison (PC) test is performed to analyse the performance of both systems. Listeners evaluated 10 sentence <without mapping, with mapping> pairs which were presented in random order of systems. Results of the PC test in terms of preference for a system are shown in Table 4. Results clearly indicate the superior quality of the system with vowel modifier mapping. Hence, in all the subsequent experiments, vowel modifiers and the corresponding vowels of a script are mapped to the same token. This mapping is also incorporated in MLCM.

Table 4: *Character-based: PC test results - with and without mapping vowels and vowel modifiers (preference in %)*

Without Mapping	**With Mapping**	**Equal**
30	51	19

4.1.2. Performance on monolingual data

Results of Section 4.1.1 are encouraging as they indicate the necessity to map similar characters across different Indian languages. In this experiment, three Indian languages (Bengali, Hindi and Malayalam) are considered. Two TTSes per language are trained with 5 hours of data using– (1) a character-to-token mapping that is specific to that script alone (language-specific) and (2) MLCM. It is to be noted that the former is a subset of the latter. These systems are evaluated through a degradation mean opinion score (DMOS) test for each language. Listeners evaluated 10 sentences per system and 5 natural recordings (i.e., 25 sentences in total) per language. The quality of the synthesised utterances is rated on a scale of 1-5 (5 being the highest) by listeners who are native speakers of the corresponding language. Evaluators are asked to rate the systems based on the naturalness of the synthesis quality. Table 5 presents the DMOS results. Results show that the use of MLCM results in a graceful degradation in performance compared to the language-specific character map.

Table 5: *Character-based: DMOS scores - monolingual data*

Language	**Language-specific**	**MLCM**
Bengali	3.60	3.37
Hindi	2.88	2.86
Malayalam	3.13	3.07

4.1.3. Performance on pooled data

This experiment investigates whether MLCM scales well when a mixture of languages is used for training. Two cases are considered– (1) pooling 5 hrs each of Bengali and Hindi (2) pooling 5 hrs each of Malayalam and Tamil. An average voice model is trained without adapting to a particular speaker. To avoid bias towards a particular speaker/language, training and validation data are selected from both datasets in a balanced manner. Further, for each case, systems are built using (a) a language-specific character map (i.e., one containing characters specific to those scripts alone) and (b) MLCM. As mentioned in Section 1, a TTS built by directly pooling different monolingual datasets does not train well. Hence, in the language-specific case too, characters that sound the same across the pooled languages are mapped together. This is also motivated by the results of Section 4.1.1. Again, the language-specific character map is a subset of MLCM.

Effectively four systems are built. These systems are evaluated through a DMOS test for each language. Only monolingual sentences are presented for synthesis. Listeners evaluated 10 sentences per system and 5 natural recordings per language. Table 6 presents the DMOS results. Results show that the use of MLCM causes a graceful degradation in performance with respect to language-specific character maps.

Table 6: *Character-based: DMOS scores - pooled data*

Language	**Language-specific**	**MLCM**
Bengali	3.37	3.27
Hindi	3.05	2.81
Malayalam	3.22	3.05
Tamil	3.12	2.88

It is observed that some of the Tamil samples generated by the Malayalam+Tamil system have a strong Malayalam accent. This may be because Tamil script has a fewer number of characters than Malayalam. In Tamil, the same character denotes both unvoiced and its voiced counterpart. For example, *ka* (unvoiced) and *ga* (voiced) are represented by the same character.

4.2. Phone-based experiments

As mentioned in Section 2.2, two types of phone-based representations are explored– transliteration and phone mapping. TTSes are trained from 5 hours each of Bengali, Hindi and Malayalam data using both these techniques. The performance of the TTSes is evaluated using DMOS tests. Listeners evaluated 10 sentences per system and 5 natural recordings per language. Table 7 presents the DMOS scores. It is observed that the performance of phone-mapped systems is better than that of transliteration systems for languages Hindi and Bengali. The synthesis quality of the phone-mapped system is slightly degraded in the case of Malayalam.

Table 7: *Phone-based: DMOS scores - transliteration vs. phone mapping*

Language	Transliterated	Phone-mapped
Bengali	3.05	3.32
Hindi	3.18	3.31
Malayalam	3.47	3.35

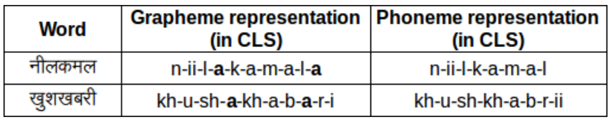
4.3. Character-based vs. phone-based experiments

The performance of character-based and phone-based TTSes is compared by a PC test. For the character-based system, MLCM is used, and for the phone-based system, the phone mapping technique is used. Listeners evaluated 10 sentence <character-based, phone-based> pairs, which were presented in random order of systems. Results of the PC test are presented in Table 8. It is seen that the performance of character-based and phone-based systems are comparable for Bengali and Malayalam, whereas the phone-based system performs better for Hindi. On closer inspection, it is observed that schwa deletion, which is the deletion of the short vowel "a", is correctly done for some words in the Hindi phone-based system. Examples of such words, along with their grapheme and phoneme representations in CLS are given in Table 9. The lack of sufficient data may also be a contributing factor for the lower performance of Hindi character-based system.

Table 8: *PC test results: MLCM vs. phone-mapped (preference in %)*

Language	MLCM	Phone-mapped	Equal
Bengali	31	29	40
Hindi	22	53	25
Malayalam	30	28	42

Table 9: *Examples of Hindi words with schwa deletion*



Word	Grapheme representation (in CLS)	Phoneme representation (in CLS)
नीलकमल	n-ii-l-**a**-k-a-m-a-l-**a**	n-ii-l-k-a-m-a-l
खुशखबरी	kh-u-sh-**a**-kh-a-b-**a**-r-i	kh-u-sh-kh-a-b-r-ii

4.4. Code-mixing

Three systems are built as part of the code-mixing experiment– monolingual Hindi, monolingual English, and their mixture (Hin+Eng). English data is the speech data of the same Hindi female speaker in the Indic TTS database [21]. As mentioned in Section 2.2, only the phone mapping technique is considered here. Monolingual systems are built using 5 hours of data, while the mixed TTS is trained using 2.5 hours each of its constituent languages. The code-mixable ability of each system is evaluated on a set of code-mixed Hindi+English text [23, 24] using a DMOS test. 10 listeners, who are proficient in both languages, evaluated 7 code-mixed synthesised speech generated by each system. Results of the DMOS test are given in Table 10. As expected, the Hin+Eng system performs the best. The Hindi system has a higher DMOS score compared to the English system. This maybe due to a higher percentage of Hindi words in the code-mixed sentences.

Table 10: *Code-mixing: Hindi and English*

System	DMOS Score
Hindi	2.83
English	2.53
Hin+Eng	3.04

4.5. Discussion

Although TTSes using MLCM are slightly degraded in performance compared to language-specific character maps, the advantage of using MLCM is that it can be directly used when more languages are pooled in. It is easily scalable to a language with a new script, as new characters just need to be mapped to the corresponding tokens. In the phone-based approach, phone mapping gives a better overall performance compared to the transliteration version. This indicates that purely phonetic based text may be better suited for low resource languages with specific G2P rules. This is also supported by the results of comparing character and phone-based approaches for Hindi. The catch is that, for a new Indian script, any language-specific G2P rule may be required to be included in the unified parser. All these experiments are conducted with 5 hours of monolingual data. Maybe with more data, such mappings (especially G2P ones) could be learnt by the network.

Most systems are observed to have low DMOS scores. Although the synthesised speech is intelligible, artifacts are introduced due to the Griffin-Lim vocoder. Replacing the Griffin-Lim vocoder by a WaveNet vocoder will result in better synthesis quality. Tacotron2 with Griffin-Lim vocoder gives a mean opinion score of 4 for a US English dataset [2]. The US English dataset is of 24.6 hours in duration while the datasets used in this work are about 5 hours in duration. It is also observed that the TTS systems sometimes produce erroneous speech, wherein parts of the utterance are repeated towards the end. To a large extent, this issue is avoided with the phone-based approach.

The synthesis samples of the conducted experiments are available at `https://www.iitm.ac.in/donlab/ssw2019/mlcm_phone/index.html`.

5. Conclusion

This paper explores character and phone-based text representations for training Indic TTSes in an end-to-end framework. Reasonably good quality TTSes are built for monolingual data and pooled data across languages using both approaches. A code-mixable TTS is also developed using the phone-mapped approach.

This is still a work in progress. But this is a stepping stone to build multilingual and code-mixable TTSes. The advantage of such a cross-lingual approach is that a generic TTS can be built and adapted to languages that are low resource, along with the ease of using end-to-end systems.

6. References

[1] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: Towards end-to-end speech synthesis," in *INTERSPEECH*, 2017, pp. 4006–4010.

[2] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R.-S. Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS Synthesis by

Conditioning WaveNet on Mel Spectrogram Predictions," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4779–4783.

[3] W. Ping, K. Peng, and J. Chen, "ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech," in *International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://arxiv.org/abs/1807.07281

[4] A. L. Thomas, A. Prakash, A. Baby, and H. A. Murthy, "Code-switching in Indic Speech Synthesisers," in *INTERSPEECH*, 2018, pp. 1948–1952.

[5] A. Prakash, J. J. Prakash, and H. A. Murthy, "Acoustic Analysis of Syllables Across Indian Languages," in *INTERSPEECH*, 2016, pp. 327–331.

[6] B. Ramani, S. Lilly Christina, G. Anushiya Rachel, V. Sherlin Solomi, M. K. Nandwana, A. Prakash, S. Aswin Shanmugam, R. Krishnan, S. Kishore, K. Samudravijaya, P. Vijayalakshmi, T. Nagarajan, and H. A. Murthy, "A common attribute based unified HTS framework for speech synthesis in Indian languages," in *Speech Synthesis Workshop (SSW)*, 2013, pp. 291–296.

[7] A. Prakash, M. R. Reddy, T. Nagarajan, and H. A. Murthy, "An approach to building language-independent text-to-speech synthesis for Indian languages," in *National Conference on Communications (NCC)*, Kanpur, India, February 2014, pp. 1–5.

[8] P. Baljekar, S. Rallabandi, and A. W. Black, "An investigation of convolution attention based models for multilingual speech synthesis of indian languages," in *INTERSPEECH*, 2018, pp. 2474–2478.

[9] E. Nachmani and L. Wolf, "Unsupervised Polyglot Text To Speech," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 7055–7059.

[10] J. Latorre, J. Lachowicz, J. Lorenzo-Trueba, T. Merritt, T. Drugman, S. Ronanki, and K. Viacheslav, "Effect of data reduction on sequence-to-sequence neural TTS," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 7075–7079.

[11] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep Voice 3: 2000-Speaker Neural Text-to-Speech," in *International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=HJtEm4p6Z

[12] K. Kastner, J. F. Santos, Y. Bengio, and A. Courville, "Representation Mixing for TTS Synthesis," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5906–5910.

[13] B. Li, Y. Zhang, T. Sainath, Y. Wu, and W. Chan, "Bytes are All You Need: End-to-End Multilingual Speech Recognition and Synthesis with Bytes," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5621–5625.

[14] Y. Cao, X. Wu, S. Liu, J. Yu, X. Li, Z. Wu, X. Liu, and H. Meng, "End-to-end Code-switched TTS with Mix of Monolingual Recordings," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6935–6939.

[15] A. Baby, N. N. L., A. L. Thomas, and H. A. Murthy, "A unified parser for developing Indian language text to speech synthesizers," in *International Conference on Text, Speech and Dialogue*, 2016, pp. 514–521.

[16] M. Naik and V. S. Chakravarthy, "A comparative study of complexity of handwritten Bharati characters with that of major Indian scripts," in *International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 3050–3057.

[17] V. C. Sekhar, V. S. Chakravarthy, and V. Pulabaigari, "An efficient Multi Lingual Optical Character Recognition system for Indian languages through use of Bharati Script," in *Document Analysis and Recognition (DAR), Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*, 2018.

[18] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala, and T. Ochiai, "Espnet: End-to-end speech processing toolkit," in *Interspeech*, 2018, pp. 2207–2211.

[19] J. Zhang, Z. Ling, and L. Dai, "Forward Attention in Sequence-To-Sequence Acoustic Modeling for Speech Synthesis," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4789–4793.

[20] D. W. Griffin, Jae, S. Lim, and S. Member, "Signal estimation from modified short-time Fourier transform," *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 32, no. 2, pp. 236–243, 1984.

[21] A. Baby, A. L. Thomas, N. N. L, and H. A. Murthy, "Resources for Indian languages," in *Community-based Building of Language Resources (International Conference on Text, Speech and Dialogue)*, 2016, pp. 37–43.

[22] S. King and V. Karaiskos, "The blizzard challenge 2013," in *Blizzard Challenge workshop*, 2013.

[23] K. Prahallad et al., "The blizzard challenge 2014," in *Blizzard Challenge workshop*, 2014.

[24] K. Prahallad et al., "The blizzard challenge 2015," in *Blizzard Challenge workshop*, 2015.