Sardin: speech-oriented text processing

Christina Tånnander[1,2], Jens Edlund[1]
[1] Speech, Music & Hearing, KTH Royal Institute of Technology, Sweden
2 MTM, Swedish Agency for Accessible Media
christina.tannander@mtm.se, edlund@speech.kth.se

Abstract

We present Sardin, a text processing system for Swedish TTS production that has recently undergone significant refactoring in preparation for public release and is soon released as free and open software. Sardin is a text processing system with the goal to prepare text for speech-centric science, such as preparing text for speech synthesis training, or for use in speech applications, for example as input of different levels and detail to different TTS systems. The current version of Sardin handles several input and output formats (EPUB, Daisy XML, generic XML, text, IPA, SAMPA), and contains modules for chunking, tokenisation, part-of-speech tagging, text normalisation, and pronunciation and prosodic information.

Introduction

Sardin is a speech-oriented text processing system. It was developed by the Swedish Agency for Accessible Media (MTM) as part of its text-to-speech (TTS) synthesis production system. The system targets long and information rich real-world texts, such as university textbooks and newspapers. It has been used in production and continuously maintained since 2007. In 2020-21, the system underwent major refactoring in preparation for an initial release as free open-source software through Språkbanken Tal (Eng. appr. *The speech bank*), a national research infrastructure for speech-centric research.

A production TTS system contains steps that are normally omitted in descriptions of TTS pipelines. Notably, there is the initial data ingestion, where the text to be read is acquired from somewhere, and a post-processing stage where presentation instructions and speech audio files are packaged and exported. In addition to plain text, Sardin can read and write several standard publication formats such as EPUB 3 (*EPUB 3.2 Final Community Group Specification*, 2019) and Daisy 3 (*Specifications for the Digital Talking Book*, 2005). This facilitates testing on real-world materials and allows the system to be used to process real-world data in preparation for training.

Traditionally, the TTS process chain is divided in two: the first step transforms the text to be read aloud to symbol sequences that are the input of the second step, which generates a speech signal based on the symbol sequence. The first step is traditionally known as the "front-end" in the TTS literature, but other terms such as "linguistic analysis" (Ebden & Sproat, 2014) are also used. This is the domain of Sardin, and we use the term *speech-oriented text processing* to highlight the fact that the process deals with text with a speech-centric view, for example looking to how it can be spoken or read aloud. We use the term *speech generation* for the second step, which can be achieved using a variety of techniques such as formant synthesis, concatenative synthesis, statistical synthesis and neural synthesis. Sardin can be configured to produce symbol sequences that are suitable for virtually any speech generation system.

TTS is increasingly becoming a standard part of our text presentation arsenal. When the European Accessibility Act (EEA) takes effect in June 2025 (EU, 2019a), the strengthened requirements on accessibility are expected to lead to an increase in TTS use by publishers. And although speech-oriented text processing is a non-trivial task, its reputation within the speech technology community has historically received little attention (Ebden & Sproat, 2014; Tran & Bui, 2021). Within the machine learning community, speech and written language alike are not research topics so much as hard tasks on which to test machine learning algorithms. The science of speech and writing are by-products in this context, and speech-oriented text processing is limited to an interest in so-called *end-to-end* TTS, where the idea is to *"to ultimately replace the whole pipeline by a single neural network predicting the audio signal corresponding to the reading of a given text"* (Perquin et al., 2020). In practice, current end-to-end systems move pronunciation modelling, and potentially some linguistic modelling, into the learned model (Watts et al., 2019), but the majority of the text processing required is not as much handled by the models as it is ignored. Tan et al., 2021 state that although *"some TTS models claim fully end-to-end synthesis"*, *"text normalisation is still needed to handle raw text with any possible non-standard formats"*. "Non-standard", here, refers in practice to any text that is not already prepared for machine learning, which is *any real-world text*. In the typical case, the machine learning literature deals with already preprocessed materials such as LJ Speech (Ito & Johnson, 2017), both for training and testing. Here, virtually all hallmarks and peculiarities of written language are absent: LJ Speech contains 19 known, explicitly enumerated abbreviations and 19 instances of non-ascii characters, and it is presented as a collection of single sentences. As an example of the somewhat artificial nature of such end-to-end processing, Tihelka et al., 2021 tested several Tacotron implementations on texts containing a small number of trivial text phenomena, with a resulting failure to speak appropriately in a majority of cases.

The text processing involves a number of more or less complex process steps:

Data ingestion. The system reads (and unpacks in the case of e.g. EPUB) the input. Often, this involves pre-normalisation steps such as converting the text to UTF8 or normalising special tokens such as spaces and different varieties of hyphen. The system then structures the text in a series of processes that may involve parsing of input formats such as HTML or XML, sentence segmentation, and tokenisation.

Text analyses. Next, a series of analyses take place. These normally use information and resources that are external to the text, such as sets of rules, dictionaries, or language models. Typical process steps include the detection of *non-standard words* (Sproat et al., 2001), that is words or expressions that need some kind of transliteration from the original writing into the sequence of words to be spoken. Typical examples include the expansion of abbreviations or the rewriting of date expressions and currencies.

Instruction. Next, the sequence of words to be spoken is embellished with structured information on *how* it should be spoken. This often involves converting the words into a phonetic representation and may also include prosodic phenomena (e.g. emphasis or speaking rate).

Generation. In the last step, the symbol sequence that is the input to the speech generation is generated. This can be done at different levels of representation, such as a series of normalised sentences of the type that the Tacotron expects as "raw" input, a series of grapheme symbols, or a series of phonetic symbols. It can also involve more complex generation, such as inserting information into an original EPUB structure in the form of SSML.

Although all these steps are found in TTS production, system descriptions generally present one of the following three scopes: (1) text normalisation descriptions from sentence segmentation or word tokenisation, including analysis, but commonly interrupted before pronunciation instruction; (2) pronunciation-oriented descriptions (e.g., grapheme-to-phoneme conversion) residing in the instructive step, or (3) speech generation. The remaining process steps, and the way in which these steps interact, are rarely mentioned.
This paper introduces Sardin, a production system for speech-oriented text processing that has recently been refactored and released under a free open-source license (Apache 2.0). Sardin is designed to be both robust and versatile. Its processing chain follows standard principles of text processing for TTS. The system utilises a large number of language dependent information resources, but its modular design makes it possible to swap out processes such as part-of-speech tagging or grapheme-to-phoneme conversion (G2P). The system permits detailed selection what information should be included in the output (e.g., pronunciations for out-of-vocabulary (OOV) words, abbreviations, law references, pause placements and durations, and level of phonetic representation) and what output format to use. Although its main purpose is to generate input for different types of speech generation, it can also be used to perform linguistic analyses or to process texts to be used as manuscripts for human speakers or as training material for neural speech generation.

Historical overview of Sardin

Sardin was originally developed in 2006, as part of the Swedish unit selection text-to-speech synthesis system Filibuster (Ericsson et al., 2007; Sjölander et al., 2008; Tånnander, 2018). Filibuster was a Swedish TTS designed to handle complex texts, namely university textbooks. It was also localised to Norwegian bokmål (2009) and Danish (2012). The commercial TTS systems at that time often had limitations to how many pronunciations user lexicons could hold, and methods for controlling the voice were less elaborate, which necessitated the development of an in-house TTS system. Since 2017, MTM uses commercial voices for Swedish and English in the production of synthetic university textbooks and for newspapers synthesised on-the-fly in the users' devices. These voices have no limitations on user lexicons, and SSML (W3C, 2010) can be used to insert pronunciations, word substitutions, pauses, inter alia. Unsurprisingly, most commercial TTS are not built for handling large amounts of complex and subject-specific text, so MTM still provides extra information to these systems, mainly by inserting phonetic pronunciations in user lexicons or as SSML into the texts representation.

Sardin

System basics

Code base

Currently most of the code base is written in Perl. Most of it is object-oriented and the development is test-driven, which facilitates regression testing. This has resulted in an extensive test battery for Swedish with more than 3,500 tests that also act as part of the system's documentation. The system is modular, and each process can be replaced by essentially anything, including modules written in other programming languages and external plug-ins. This increases its usefulness for other languages, including languages unrelated to Swedish and other Germanic languages, although there would be less code re-use for languages with significantly different writing systems, such as Japanese. Sardin was originally developed for Swedish and has then been localised to English, although on a smaller scale.

Information slots

Sardin has an extendible number of information slots attached to each token and chunk. Currently implemented token slots include ***orth*** (the original orthography of the token), ***pos*** (part-of-speech and morphological information), ***exprType*** (roughly corresponding to what in the literature is referred to as semiotic classes), ***exp*** (the expanded form of the orthography), ***pron*** (phonetic transcription of some detail), ***pause*** (pause durations), ***decomp*** (compound decomposition), ***orig*** (information about the origin of the pronunciation: dictionary, automatic compound, CART tree etc.), and ***SSML*** (output SSML for TTS control). A slot for ***markup*** is being implemented, holding information retrieved from the original input document, for example structural information such as headings `(<h1>`, `<h2>` etc.), list elements (`<li>`), page numbers (`<pagenum>`), and print style information such as emphasis (`<em>` and `<strong>`, typically realised as italics and bold face).

Information sources

Several resources are used by the Sardin modules, most of them language dependent. The pronunciation dictionaries constitute a significant resource with information such as phonetic pronunciation, part-of-speech tag, morphological information, and language linked to the orthography. Another example is the list of abbreviations, which contain five fields: alternative orthographies; alternative expansions; expansion rules for abbreviations that can take different expansions; a flag showing if the abbreviation can occur in sentence final position; and a flag showing if the abbreviation is case sensitive or not. Other information resources are lists of acronyms, compound parts with their corresponding pronunciations (used for decomposing compounds and creating pronunciations for them), and clusters of vowels and consonants that are accepted in the language in question (for checking if an orthographic string is pronounceable). Finally, there are a several sets of trigger words, for example words signalling that a number expression to the right or the word is a range (e.g., “chapter” or “between”), or a Roman numeral (e.g. “part”).

Data ingestion

Reading/unpacking

Sardin takes an annotated (XML/EPUB3) or a plain text as input. The document is parsed and the text content and information about the *ML markup sent to the next module. At this stage, we do not make any changes in the text (e.g., normalising spaces or hyphens), since we must be able to present the text *exactly* as it appears in the input document after enriching it with for example SSML tags.

Chunking

The text is then chunked into manageable sizes, in its current form into sentences. To avoid incorrect sentence splits at periods belonging to abbreviation, a temporary abbreviation markup is done before the splitting. Briefly described, the text is first split at major delimiters and then over-generated splittings are removed, for example at name initials or before said phrases (*“Give it to me! he shouted.”*).

Tokenisation

Each sentence is then sent to the tokeniser. Again, abbreviations need special treatment, and the first step is to unify them as one single token along with their belonging periods. Next the text is split at spaces and delimiters, such as commas, quotes and brackets. There is a range of rules, for example splitting at hyphens between digits (but not between letters) and merging numbers with thousand separators (space in Swedish, usually comma in English).

Analyses and enrichment

The linguistic analyses include part-of-speech tagging, and text normalisation classification and expansion,

Part-of-speech tagging

Each sentence is then sent to the part-of-speech tagger, which is a statistical tagger based on unigram, bigram and trigram probabilities, and complemented with rules where statistics are known to fail.

Text normalisation, classification

The analysis/markup module assigns each token to a class, in line with Sproat et al., 2001. Table 1 lists the classes in the order their rules are applied and gives examples of tokens that belong to each class, the expected expansion (which is performed in the following expansion module) and a translation of the expansion to English for the reader’s convenience. References are classified first to save time, since otherwise some of the following classifications would apply on parts of the references. Currently, all classifications are rule-based. Note that the class set is not fixed: classes can be added and used by the succeeding processes.

Text normalisation, expansion

At the time of writing, Sardin contains four expansion modules: abbreviations, characters, numbers and references. As an example of **abbreviation** processing, “sek.” or “sek” can be expanded to “sekund” (Eng. second) or “sekunder” (Eng. seconds) depending on the digit to the left of the abbreviation, but in sentence-final position, case sensitive “SEK” denotes the currency code for Swedish crown.
Characters (e.g. daggers, hyphens and at-signs) all have their own expansion rule sets. Hyphens, for example, are expanded to “till” (to) if it belongs to the Interval class, to itself if it belongs to the Date class (and not to Interval), to “streck” (dash) if it occurs between digits that aren’t part of an interval, and to pause if no other rule applies. The **numeral** expansion module expands numerals to cardinals, ordinals, or to single digits. If they belong to the Year class, they are expanded as such, and Roman numerals are converted into Arabic. Finally, the **reference** expansion takes care of law references. This is a complicated procedure, since the references need to be parsed to chunk the numbers with the correct unit (chapter, section, part etc.). For instance, “12 kap. 19 §” expands straightforwardly to “tolfte kapitlet nittonde paragrafen” (twelfth chapter nineteenth section), but more complex structures are error prone and difficult to read out even for a human. “12a-13b kap.” could be expanded to “twelfth a to thirteenth b chapter”, which is hard to understand. Instead all law references are converted into a unit:numeral format: “kapitel tolv a till tretton b” (chapter twelve a to thirteen b).

Pronunciation

In the Pronunciation module, each (potentially expanded) token receives its pronunciation in the following manner: First, the domains of electronic addresses are given their pronunciations (for example, ‘se’ marked as Email or URL is assigned a spelled pronunciation); then acronyms are given their pronunciation from a dictionary or assigned an automatic (spelled out) pronunciation. Next, a dictionary lookup takes place, fetching pre-transcribed pronunciations and other information of the words. The lookup function uses the part-of-speech information to disambiguate homographs such as *record* (noun or verb). If the word is not found in any dictionary, an affix check takes place, checking if the word is an inflected known word (e.g., “record-s”) in which case the pronunciation of the known word is modified accordingly, then used. Next comes a compound check, where a compound decomposer checks if the word is built up by known compound parts. If so, automatic methods for compound pronunciation are used. The pronunciations of the known words are concatenated, and the stress pattern changed according to language dependent compound stress rules. Finally, if none of the above methods could be used, the word is checked for its pronounceability. If the word is made up of consonants only, or if it contains graphemic clusters that are not part of Swedish

Class	Input example	Swedish expansion	Translation
Reference	3 kap. 2-3 st. 4§	kapitel tre stycke två till tre paragraf fyra	Chapter three part two to three section four
Abbreviation	En s. k. elefant.	En så kallad elefant.	A so called elephant.
Initials	P.J. Harvey	P J Harvey	P J Harvey
Date	1/3-1951	Första i tredje nitton-hundra-femtio-ett	First in third nineteen hundred fifty-one
Time	Kl. 19.15	Klockan nitton och femton	Clock nineteen and fifteen
Email	p.j@harvey.com	P punkt J snabel-a harvey punkt com	P dot J at harvey dot com
URL	www.kth.se	V V V punkt K T H punkt S E	W W W dot K T H dot S E
Filename	C:/myfile.txt	C kolon snedstreck myfile punkt T X T	C colon slash myfile dot T X T
Roman number	Sidan XII	sidan tolv	page twelve
Acronym	KTH	K T H	K T H
Decimal	3,14	Tre komma fjorton	Three comma fourteen
Phone number	08-12 12 12	Noll åtta streck tolv tolv tolv	Oh eight dash twelve twelve twelve
Ordinal	31 mars	Trettio-första mars	Thirty-first March
Year	Sproat (1996)	Sproat (nitton-hundra-nittio-sex)	Sproat (nineteen hundred ninety-six)
Interval	Kapitel 5-8	Kapitel fem till åtta	Chapter five to eight
Currency	£5,80	Fem pund och åttio cent	Five pounds and eighty cents

Table 1. Examples of classes and expansions.

orthotactics, they are spelled out. If it can be pronounced, they are sent to a G2P converter, which uses a CART (classification and regression) tree to produce an automatic pronunciation.

Next, the Pause and Prosody module assigns pauses of different lengths according to their TN markup and/or orthography, for example at certain dashes or hyphens, minor and major delimiters. There is also the embryo of an emphasis assignment process – a simple rule that assigns emphasis to tokens that were marked with `<em>` or `<strong>` (if surrounded by unmarked tokens). Depending on the capabilities of prosody control in the synthesiser, this feature may or may not be realised in the output speech stream.

Output and packaging

In the final step, the output of the system is generated. The process is flexible, and the output can be produced in a variety of formats and packaging. The internal phonetic symbols used by Sardin are converted into the desired target format, for example IPA or SAMPA, using a symbol conversion table and a handful of rules, enabling the preparation of correct input to the subsequent speech generation. It is also possible to produce SSML output, and to choose which features to include based on what the TTS is able to handle. Knowing your TTS voice and the mistakes it makes is essential to support it with appropriate guidance, and to avoid foisting it with unnecessary information. The following choices are currently available: `<phoneme>` (pronunciations): all tokens; OOV tokens, acronyms; name initials; email addresses; URLs; filenames; `<sub>` (substitutions): abbreviations; numerals; dates; time expressions; years; currencies; decimals; phone numbers; ordinals; intervals; fractions; `<break>` (pauses): all pauses; and Other (mixed SSML tags): law references; page references; hyphens; interval hyphens. The original document is then rebuilt with its original markup and the SSML markup of your choice. Alternatively, the output can be produced sentence for sentence, for example in some format that is used for training and testing a TTS engine.

Quality assurance

Sardin is a full system handling the entire process from XML parsing to SSML insertions. The analysis module (TN) can be evaluated with methods such as those used by Flint et al. (2017), Reichel & Pfitzinger (2006) and Tyagi et al. (2021), and the automatic G2P module can be compared to other G2P systems. However, there is no TN test set for Swedish to our knowledge, and the English version of Sardin is not sufficiently elaborated to warrant testing. On the other hand, the test-driven development of Sardin ensures that the system is suitable for its primary purpose, and the fact that Sardin has been used in real-world production of synthetic speech for more than a decade also speaks of its capabilities.

Conclusion and future work

The aim of this paper is to present the basics of Sardin in light of the field of speech-oriented text processing, as part of its release as publicly available free and open software[1]. The refactoring and open sourcing of version 1.0 did not include any modernisation or improvement of text processing methods or of the capacity to handle different types of text, but it did lead to efficiency gains and a system that is considerably easier to adapt and extend.

Concerning the usefulness of Sardin, we know that data-driven methods for text processing create unacceptable errors, and researchers working with these methods commonly add rule-based pre- and/or post-processing both for text-to-speech (e.g. Reichel & Pfitzinger, 2006; Tyagi et al., 2021) and for speech-to-text (Tran & Bui, 2021). So there is still a need for preparing text before speech generation for production purposes, which is the current main use of Sardin: to unpack, process, enrich (according to what the speech generation system can handle), and rebuild Daisy XML, EPUB or text files. A key strength of Sardin is the ability to add information slots and resources for specific purposes – to allow the use of whatever information is available: information about print style, domain, inter alia. But with a little adaptation, Sardin can be used for other purposes, for example: for text processing of STT/ASR results, for sentence

[1] https://www.sprakbanken.speech.kth.se/software/sardin/

segmentation for synchronisation of text and speech, word tokenisation for frequency calculations, and pronunciation generation for user lexicons of specific TTS voices. Perhaps most importantly, the system can be used to create test and training data for TTS and STT/ASR in a repeatable and transparent manner, which would benefit the ML world. We could for example set up a configuration that can go from the original texts contained in LJ Speech to (a) manuscripts for reading aloud and (b) test and training materials (the current LJ Speech distribution). Among other things, this would make it a lot easier to retrain LJ Speech-based implementations on new data – something can be a struggle when LJ Speech dependencies are built in but undocumented. A clear drawback is that Sardin is not language independent. Adapting the system to a new language requires several steps, including the creation of new lists of abbreviations, acronyms and pronunciations, language-dependent part-of-speech tagging, adaptation of expansion rules, G2P etc.

In next steps, we will continue with research and evidence-based development of speech-oriented text processing, including fine-tuning of existing modules (e.g. year detection), replacement of some processes with more modern techniques (e.g., G2P and part-of-speech tagger), and introduction of new functions (e.g. data-driven text classification). We hope that the Swedish interpretation of the Directive on Copyright in the Digital Single Market (EU, 2019b) will give us better access to sufficient amounts of training data for the text domains we usually deal with: university text books and news, so we can train models for additional models without violating the copyright law. We can then use methods such as those described in Sproat & Jaitly (2017), creating a normalised corpus with the mostly rule-based version of Sardin today to use as reference when testing modified Sardin versions or external systems.

Acknowledgements

This work is funded partly by the Vinnova project Deep learning-based speech synthesis for reading aloud of lengthy and information rich texts in Swedish (VINNOVA2018-02427), and partly by the Swedish Agency for Accessible Media (MTM). The results will be made more widely accessible through the Swedish Research Council funded national infrastructure Språkbanken Tal (VR2017-00626).

References

Ebden, P., & Sproat, R. (2014). The Kestrel TTS text normalization system. *Natural Language Engineering, 21*(3), 333–353. https://doi.org/10.1017/S1351324914000175

Ericsson, C., Klein, J., Sjölander, K., & Sönnebo, L. (2007). Filibuster – a new Swedish text-to-speech system. Proceedings of Fonetik 2007, 50, 33–36. KTH-TMH.

EU. (2019a). On the accessibility requirements for products and services (Directive No. 2019/882; p. 114). European Parliament and the Council of the European Union. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32019L0882&from=EN

EU. (2019b). Directive (EU) 2019/790 of the European Parliament and of the Council of 17 April 2019 on copyright and related rights in the Digital Single Market and amending Directives 96/9/EC and 2001/29/EC. file:///C:/Users/christina.tannander@mtm.se/Zotero/storage/EEJJ4UVD/DIRECTIVE%20%20(EU)%20%202019%20%20790%20%20OF%20%20THE%20%20EUROPEAN%20%20PA.pdf

Flint, E., Ford, E., Thomas, O., Caines, A., & Buttery, P. (2017). A text normalisation system for non-standard English words. Proc. of the 3rd Workshop on Noisy User-Generated Text, 107–115. https://doi.org/10.18653/v1/W17-4414

Garrish, M., & Cramer, D. (Eds.). (2019). EPUB 3.2 final community group specification (W3C Community Group Specification) [Specification]. W3C. https://www.w3.org/publishing/epub3/epub-spec.html

Ito, K., & Johnson, L. (2017). The LJ Speech dataset. https://keithito.com/LJ-Speech-Dataset/

Perquin, A., Cooper, E., & Yamagishi, J. (2020). Grapheme or phoneme? An analysis of Tacotron's embedded representations (ArXiv 2010.10694). http://arxiv.org/abs/2010.10694

Reichel, U. D., & Pfitzinger, H. R. (2006). Text preprocessing for speech synthesis. Proc. of the TC-STAR Workshop on Speech-to-Speech Translation, 207–212.

Sjölander, K., Tånnander, C., & Sönnebo, L. (2008). Recent advancements in the Filibuster text-to-speech system. Proc. of SLTC 2008.

Specifications for the digital talking book (Specification Z39.86-2005 (R2012)). (2005). The DAISY Consortium. https://daisy.org/activities/standards/daisy/daisy-3/z39-86-2005-r2012-specifications-for-the-digital-talking-book/

Sproat, R., Black, A. W., Chen, S., Kumar, S., Ostendorf, M., & Richards, C. (2001). Normalization of non-standard words. Computer Speech and Language, 15, 287–333. https://doi.org/10.1006/csla.2001.0169

Sproat, R., & Jaitly, N. (2017). RNN approaches to text normalization: A challenge (https://arxiv.org/abs/1611.00068; ArXiv 1611.00068). arXiv.

Tan, X., Qin, T., Soong, F., & Liu, T.-Y. (2021). A survey on neural speech synthesis. Preprint ArXiv:2106.15561 [Cs, Eess], 63.

Tånnander, C. (2018). Speech synthesis and evaluation at MTM. Proc. of Fonetik 2018, 75–80.

Tihelka, D., Matoušek, J., & Tihelková, A. (2021). How much end-to-end is Tacotron 2 end-to-end TTS system? In K. Ekštein, F. Pártl, & M. Konopík (Eds.), Procs. Of TSD 2021 (pp. 511–522). Springer International Publishing. https://doi.org/10.1007/978-3-030-83527-9_44

Tran, O. T., & Bui, V. T. (2021). Neural text normalization in speech-to-text systems with rich features. Applied Artificial Intelligence, 35(3), 193–205. https://doi.org/10.1080/08839514.2020.1842108

Tyagi, S., Bonafonte, A., Lorenzo-Trueba, J., & Latorre, J. (2021). Proteno: Text normalization with limited data for fast deployment in text to speech systems. Proc. of NAACL 2021, 72–79. https://doi.org/10.18653/v1/2021.naacl-industry.10

W3C. (2010). Speech Synthesis Markup Language (SSML) Version 1.1 (W3C Recommendations) [Specification]. W3C; W3C. https://www.w3.org/TR/speech-synthesis11/

Watts, O., Eje Henter, G., Fong, J., & Valentini-Botinhao, C. (2019). Where do the improvements come from in sequence-to-sequence neural TTS? Procs. of SSW *10),* 217–222. https://doi.org/10.21437/SSW.2019-39