VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master’s thesis

Work Title in English

Work Title in Lithuanian

Aleksandr Jan Smoliakov

Supervisor	:	Gerda Ana Melnik-Leroy
Scientific advisor	:	pedagogical/scientific title Name Surname
Reviewer	:	pedagogical/scientific title Name Surname

Vilnius
2025

Contents

Introduction

Text-to-Speech (TTS) synthesis, also known as speech synthesis, is the process of converting written text into human-like spoken words. Nowadays TTS is a key technology in various applications, including virtual assistants, accessibility tools, and language learning platforms.

The goal of creating machines that can speak like humans has captivated researchers for centuries. One of the earliest known attempts dates back to the 18th century, with Wolfgang von Kempelen’s mechanical speech machine that utilized a bellows-driven lung and physical models of the tongue and lips.

Over the centuries, advancements in technology and understanding of human speech have driven significant progress in this field. Today’s state-of-the-art systems, dominated by end-to-end (E2E) neural models, have achieved highly naturalistic speech with unprecedented acoustic quality. Notably, these end-to-end systems have unified the entire synthesis process into a single neural network, eliminating the need for complex multi-stage pipelines.

However, this success has come with new dependencies and failure modes that force us to reconsider how input should be incorporated into the synthesis pipeline.

First of all, E2E models rely on a large amount of high-quality training data to achieve optimal performance. This poses challenges, particularly for low-resource languages or domains where such data is scarce.

Additionally, these models often struggle with numbers, dates, abbreviations, and other non-standard words (NSWs) that are common in real-world text but not well represented in training data. This can lead to mispronunciations or unnatural prosody, undermining the overall quality of the synthesized speech.

To address these issues, modern TTS systems still benefit from explicit linguistic knowledge, such as phonetic transcriptions or text normalization (TN) rules, to handle these challenging cases effectively.

This thesis aims to explore the integration of text normalization techniques into end-to-end TTS systems, with a focus on improving the objective and subjective quality of synthesized speech for a low-resource language – Lithuanian.

While the necessity of TN is acknowledged, the specific impact of different TN strategies on E2E model performance for morphologically rich, low-resource languages like Lithuanian remains underexplored.

The research questions this thesis seeks to answer include:

- What impact does Lithuanian text normalization have on the quality of synthesized speech in popular end-to-end TTS systems?
- What is the trade-off between the complexity of text normalization rules and the performance of end-to-end TTS models?

TODO Structure of the thesis

1 Literature review

Before diving into the specifics of text normalization, it is important to establish the broader context of TTS research and its evolution over time.

1.1 Text-to-speech synthesis

1.1.1 Early methods of speech synthesis

The first known attempts to create artificial speech date back to the late 18th century. These early devices were purely mechanical, relying on physical models to mimic the human vocal tract. One notable example is Wolfgang von Kempelen's invention called the "Acoustic-Mechanical Speech Machine", which utilized a bellows-driven lung and physical models of the tongue and lips to articulate both vowels and consonants. Such early efforts had limited success, being able to produce only individual phonemes or simple syllables.

The advancement of electricity and electronics in the late 19th and early 20th centuries opened new possibilities for speech synthesis. Homer Dudley's Voder (Voice Operation Demonstrator), unveiled in 1937, represented the first successful electronic recreation of human speech. This landmark invention allowed a skilled human operator to synthesize intelligible speech by manipulating a complex arrangement of keyboards and pedals, each controlling different acoustic parameters of speech production. The Voder marked a shift from mechanical to electronic synthesis methods.

In the decades that followed, two main approaches for speech synthesis emerged: concatenative synthesis and parametric synthesis.

1.1.2 Concatenative and parametric TTS

The concatenative synthesis approach synthesizes speech by piecing together pre-recorded segments of human speech. This method involves several steps. First, it requires pre-recording a large database of speech segments spoken by a human voice actor.3 Each segment is labeled and indexed based on its phonetic and prosodic properties.

During synthesis, the system breaks down the input text into short linguistic units (such as phonemes or syllables) using a text analysis module. Then, it queries the speech database to find the best-matching segments for each unit. The retrieved segments are blended and concatenated to form a continuous speech waveform. The system may uses signal processing techniques to smooth the transitions between segments and adjust pitch and duration to match the desired output characteristics.

Concatenative synthesis can produce natural-sounding individual speech units, but the final audio often has noticeable glitches and breaks at the points where segments are joined together. The segments may not blend smoothly due to differences in pitch, duration, and timbre. The prosody also tends to sound "choppy" and unnatural, since stringing disjointed segments together does not capture the natural rhythm and intonation patterns of connected speech.5

Finally, concatenative synthesis requires language-specific expertise to design and maintain the underlying speech database and selection algorithms. This need for extensive data can make it challenging to develop concatenative TTS systems for low-resource languages or dialects.

In contrast, statistical parametric speech synthesis (SPSS) uses statistical models, typically Hidden Markov Models (HMMs), to generate the parameters that control a speech waveform.

This method involves training a statistical model on a large corpus of recorded speech. The model learns the relationship between linguistic features (like phonemes and prosody) and the acoustic features of the speech signal, such as spectral envelope and fundamental frequency. During synthesis, the system takes text as input, converts it to a sequence of linguistic features, and then uses the trained model to generate a corresponding sequence of acoustic parameters.

Compared to concatenative synthesis, the statistical approach allows for more flexibility and control over the speech synthesis process, enabling the generation of a wider variety of voices and speaking styles. However, HMM-based synthesis had a persistent problem: the statistical averaging built into the models tended to over-smooth the acoustic features, creating a characteristic "buzzy" or "muffled" sound that lacked the sharpness and detail of natural human speech.2

1.2 Neural network-based TTS

The limitations of these complex, multi-stage pipelines opened the door for a new approach, the end-to-end (E2E) model. E2E systems learn the entire speech synthesis process–from input text directly to acoustic output–using a single neural network.8 This approach promised to eliminate the need for hand-crafted pipelines that were difficult to design, required extensive expertise, and suffered from errors that accumulated across multiple components.10 By learning directly from text-audio pairs, E2E models showed they could produce speech with higher naturalness and expressiveness than previous methods, representing a significant leap in TTS technology.9

1.2.1 Architecture of End-to-End TTS models

The main architecture for modern E2E TTS is the sequence-to-sequence (seq2seq) framework, known through e.g. Google's Tacotron model.10 This framework has three main parts: an encoder, an attention mechanism, and a decoder.8 The encoder–usually a recurrent neural network (RNN) or a more complex module like the CBHG (Convolution Bank + Highway network + Gated Recurrent Unit)–processes the input text and converts it into a high-level representation.11 The attention mechanism acts as a bridge, learning to align the text representation with the output audio frames. At each step, it decides which part of the input text the model should focus on to generate the next piece of audio.8 The decoder, also typically an RNN, takes the encoder output and uses the attention context to generate an acoustic representation of the speech, one frame at a time.8

An important design choice in models like Tacotron [**tacotron**] is that the decoder doesn't directly create the final audio waveform. Instead, it predicts an intermediate acoustic representation, typically a mel-spectrogram.6

TODO explain mel-spectrogram in a previous section

1.2.2 Popular E2E models

TODO Tacotron 2, FastSpeech 2, VITS

1.2.3 Limitations of E2E models

While E2E models have achieved impressive results, they also bring new challenges. These models work like “black boxes” - they produce high-quality speech, but at the cost of transparency, robustness, and data efficiency. A major challenge is their significant data requirements. Training a good E2E TTS model typically needs tens or hundreds of hours of professionally recorded and transcribed speech from a single speaker.8 Many of the world’s languages may not have this much data available, which makes E2E approaches difficult to use for low-resource languages.16

These models can also be quite fragile. Attention-based seq2seq models often struggle when they encounter inputs that are very different from their training data, like unusually long sentences or text from unfamiliar topics.18 When this happens, the model can fail catastrophically - it might skip words, repeat them, or produce meaningless noise. This shows that the model’s alignment mechanism is brittle and doesn’t really understand language in a deep way.19 Even when working properly, neural models can suffer from ”over-smoothing.” Similarly to HMM-based systems, they learn statistical averages from training data, which can make the speech sound clear but lacking in natural variation and detail - essentially producing an “averaged” sound that feels less dynamic than human speech.20

Perhaps most importantly for this review, the ”end-to-end” label is somewhat misleading. E2E models are trained on clean, normalized text and cannot directly process raw text as it appears in the real world. They tend to struggle with numbers, dates, currencies, abbreviations, and other symbols that aren’t spelled out as words.1 This means that every modern E2E TTS system still requires a text pre-processing function to handle Text Normalization (TN).

This function takes raw input text and converts it into a fully spelled-out form that the E2E model can work with. The continued need for this function reveals an important limitation: while we have successfully unified acoustic modeling into a single network, the challenge of interpreting written language remains a separate problem that must be addressed beforehand. Rather than eliminating the text processing pipeline, the E2E approach has simply moved this complexity to a preprocessing step that sits outside the main model.

1.3 Text normalization

Text Normalization (TN) is a fundamental preprocessing step in the TTS pipeline. It serves as a translator between written text as it appears in the real world–with all its numbers, abbreviations, and symbols–and the clean, word-based form that TTS systems need to produce natural speech. Without proper normalization, a TTS system will struggle with a large portion of everyday text, producing poor or incorrect pronunciations.

There are two main approaches to TN systems: rule-based systems that rely on hand-crafted rules, and data-driven machine learning models. The choice depends on how much data is available, how complex the language is, and how much tolerance there is for serious errors that completely change the meaning.

The main job of Text Normalization is to find and convert non-standard words (NSWs)–tokens that are written differently from how they are spoken–into their proper spoken form.22 This step must happen before the TTS system can convert letters to sounds or process character sequences.3 NSWs are found in every language but take different forms. Common examples in English include numbers ("10kg" → "ten kilograms"), dates ("Jul 5th" → "july fifth"), monetary amounts ("$24.12" → "twenty four dollars twelve cents"), abbreviations ("St." → "Saint" or "Street"), and various symbols.26

Most TN systems work through three main stages. First is tokenization, where the input text gets split into individual words or meaningful pieces, typically using spaces and punctuation as boundaries.23 Second is classification, where each piece is examined to determine whether it's a regular word or an NSW. If it's an NSW, the system assigns it to a specific category like CARDINAL, DATE, TIME, or MONEY.23 This classification step is important because how something gets spoken depends entirely on its category. The final step is verbalization, where the classified NSW gets expanded into its full spoken form.23

One significant challenge in this process is handling context. Most NSWs cannot be normalized using a simple lookup table–the correct spoken form often depends on the surrounding words and sentence structure. A clear example in English is the abbreviation "St.", which should be "Saint" in "St. John" but "Street" in "12 St. John St.".1 Similarly, the string "1/4" could be spoken as "one fourth", "January fourth", or "April first", and only the context can tell us which is correct.22 This means an effective TN system must analyze not just the target token, but also the words around it to make the right choice.

1.3.1 Methodologies for text normalization

Two primary methodologies have been developed to address the TN problem: rule-based systems and data-driven models.

Traditionally, TN is handled by rule-based engines. These systems rely on a set of handcrafted rules, often implemented using cascades of regular expressions or.4 In a TN system, an input string is mapped to an output string through a graph-based grammar, allowing for complex and context-sensitive transformations. The primary strength of rule-based systems is their precision and controllability. Their behavior is deterministic, and for well-defined semiotic classes, they can achieve very high accuracy.32 Because they encode linguistic knowledge directly into rules, they do not require large corpora of parallel normalized text for training, making them particularly well-suited for low-resource languages.4 However, their main drawback is the immense human effort required for their creation and maintenance. Developing a comprehensive set of rules for a single language can involve thousands of individual grammars, demands deep linguistic expertise, and is a labor-intensive process.33 .21

An alternative approach treats TN as a machine learning problem, typically framing it as a sequence-to-sequence task analogous to machine translation.30 In this setting, a model learns the mapping from an unnormalized text sequence to its verbalized form directly from a large corpus of examples. Various architectures have been explored, including Recurrent Neural Networks (RNNs) 23, full sequence-to-sequence models with attention 30, and, more recently, Large Language Models (LLMs) used in a few-shot prompting scenario.22 The principal advantage of these models is their ability to learn complex, subtle, and non-linear patterns from data without needing explicitly programmed rules. This can allow them to generalize better to unseen inputs and potentially capture nuances that are difficult to encode in a formal grammar.35 However, their primary weakness is their dependence on large-scale, high-quality parallel training data (i.e., pairs of unnormalized and normalized text), which is extremely scarce and difficult to create.24 This data bottleneck is a major obstacle to applying purely data-driven TN to low-resource languages.

1.3.2 Text normalization in Lithuanian

Research into Lithuanian text normalization has focused on rule-based methods that can explicitly encode linguistic knowledge. In their work, Kasparaitis has developed a comprehensive TN system for Lithuanian that uses regular expressions to identify and verbalize NSWs.23 A key contribution of this work is the development of a detailed taxonomy of semiotic classes specifically adapted to the nuances of the Lithuanian language. This taxonomy provides a structured framework for the normalization task, outlining the specific categories of NSWs that a system must handle. The table below presents a selection of these semiotic classes, illustrating the breadth of the normalization challenge in Lithuanian.

Code	**Explanation**	**Examples**	**Source**
EXPN	Expansion (abbreviations)	liet. → lietuviškai	23
LSEQ	Letter sequence (acronyms)	VU → vė-u	23
ASWD	Read as word	KAM	23
NUM	Cardinal number	10 km.	23
NORD	Ordinal number	XIX a., 1941-ųjų	23
NTEL	Telephone	8-611-99999	23
NTIME	Time	10:45	23
NDATE	Date	2018 m. spalio 15 d.	23
NYEAR	Year	1941 m.	23
URL	URL, email	pkasparaitis@yahoo.com	23

1 table. *Semiotic classes for Lithuanian text normalization*

This taxonomy highlights the necessity of a system that can handle everything from common abbreviations (EXPN) to various numerical expressions (NUM, NORD, NTEL, etc.). Crucially, the verbalization of many of these classes, particularly NUM and NORD, will require the generation of correctly inflected forms based on the surrounding syntactic context, a challenge that lies at the heart of this thesis. The existing body of work on Lithuanian TTS and TN points toward an unavoidable conclusion: a high-quality E2E synthesis system for the language cannot function as a simple, monolithic black box. It is critically dependent on a sophisticated frontend that performs at least two major linguistic processing tasks: first, a TN module to correctly verbalize NSWs into their proper morphological forms, and second, an accentuation module to insert the stress and tone markers that guide the E2E model’s prosody. The frontend is not an optional extra; it is an indispensable component that encodes the deep, rule-governed linguistic knowledge that the neural backend is incapable of learning on its own.

1.4 Morphologically rich languages

2 Methodology

2.1 Datasets

2.1.1 Common Voice dataset

2.1.2 Liepa 2 dataset

2.2 Text normalization methods

2.3 Model architecture and training

2.3.1 Model architecture

2.3.2 Experimental setup

2.3.3 Training details

2.4 Evaluation protocol

3 Results

3.1 Quantitative results

3.2 Qualitative results and analysis

4 Discussion

4.1 Interpretation of results

4.2 Error analysis

4.3 Comparison with existing works

4.4 Limitations of the study

5 Conclusions and future work

5.1 Summary of findings

5.2 Conclusions

5.3 Future work

6 References – FIXME not using proper citation style yet