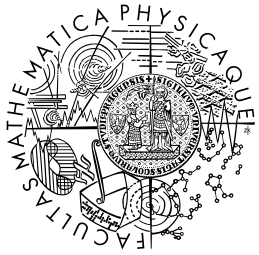


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jakub Růžička

Normalization of numbers into spoken form for text-to-speech systems

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: Mgr. et Mgr. Ondřej Dušek, Ph.D.
Study programme: Informatika
Study branch: IPP2

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague 27.4.2023 .....

Author's signature

Firstly, I would like to thank my supervisor Mgr. et Mgr. Ondřej Dušek, Ph.D., for all his help and patience with the theoretical part of the thesis. Furthermore, RNDr. Jan Cuřín, Ph.D., Ing. Josef Opička, Ph.D., and Ing. Petr Fousek, Ph.D., for their assistance with the practical part of the thesis. Finally, also to my family and other annotators, thanks to whom I was able to evaluate my work.

Title: Normalization of numbers into spoken form for text-to-speech systems

Author: Jakub Růžička

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., Institute of Formal and Applied Linguistics

Abstract: A necessary part of any text-to-speech system is the normalization of numbers and words containing numbers. The accuracy of this process can significantly affect the quality of the resulting speech. The main goal of this work is the design and implementation of a number normalization module for Czech. Words containing digits are first assigned to one of the predefined categories. Based on the category given, possible spoken forms are subsequently generated. For the selection of the contextually correct variant, an existing language model is used. The system is distributed as a Python package and can run on Linux or in a Docker container whose configuration is part of the project. Moreover, a specialized data annotation application has been designed and written for creating the datasets for the Czech text normalization task. Two datasets with 1,882 sentences and 3,185 words requiring normalization were obtained using the data annotation service. The system achieved a sentence-level accuracy of over 80% on both datasets. We perform a detailed error analysis based on the results, and propose further improvements.

Keywords: Czech text normalization, number normalization, text-to-speech systems, weighted finite-state transducer

Název práce: Normalizace čísel pro výslovnost syntézou řeči

Autor: Jakub Růžička

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. et Mgr. Ondřej Dušek, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Nezbytnou součástí každého systému pro syntézu řeči je normalizace slov obsahujících číslice. Přesnost tohoto procesu může významně ovlivnit kvalitu výsledné promluvy. Hlavním cílem této práce je návrh a implementace modulu pro normalizaci číslic v češtině. Slova obsahující číslice jsou nejprve zařazena do jedné z předem definovaných kategorií. Na jejímž základě jsou následně generovány možné rozepsané podoby. Pro výběr kontextově správné varianty je využit existující jazykový model. Systém je distribuován jako balíček v jazyce Python. Může běžet na systému Linux nebo v Docker kontejneru, jehož konfigurace je součástí projektu. Současně byla navržena a napsána specializovaná aplikace pro anotaci dat, která slouží k vytváření datových sad pro normalizaci textu. Pomocí aplikace byly vytvořeny dvě datové sady s 1.882 větami a 3.185 slovy vyžadujícími normalizaci. Systém dosáhl na obou získaných datasetech přesnosti přes 80 % (měřeno na úrovni věty). Na základě analýzy chyb popisujeme možná zlepšení.

Klíčová slova: normalizace českého textu, normalizace čísel, systém pro syntézu řeči, vážený konečný převodník

Contents

Introduction

Speech is a natural way of expressing our needs and receiving information. Today we are taking advantage of the progress achieved in natural language processing and increasingly communicate with our devices using voice. We can ask our devices to initiate a phone call while performing other tasks, such as driving a car or cooking lunch.

We say our request, and the voice assistant processes it and responds to us. This interaction, belonging to the domain of dialogue systems, has three main parts: Processing the user input, resolving the request, and answering the user. For more on this topic, we recommend McTear (2020).

This work will focus on the part where the assistant responds, specifically on preparing the message to be read out by a speech synthesizer. Today the responses from the assistants seem very natural to us because of significant advances in neural network (NN) models used for text-to-speech (TTS) synthesis (Kaur and Singh, 2022).

The first part of a TTS pipeline – text preparation or text preprocessing – includes, among other things, normalization of text. One of the main tasks of normalization is the transformation of abbreviations and numbers into their spoken form (Zhang et al., 2019). This process removes many special cases a synthesizer would struggle to deal with. For example, `42` is expanded to `forty-two`; therefore, the synthesizer can read it as a regular word. The task has an additional challenge in morphologically rich languages such as Czech, where the correct form depends on context. See Table 1.2 for examples.

This thesis aims to implement and evaluate a module for generating numbers for Czech in spoken form. The scope of the thesis also includes complex forms containing numbers such as dates. The development is done in collaboration with The Mama AI,[1] where the module will be used in their TTS system. The Mama AI provided the public API design for our module and supported the author with countless consultations. Nevertheless, the herein described implementation is purely the work of the author.

We set the goal to create a Czech normalization module for at least cardinal numbers, ordinal numbers, decimal numbers, dates, times, measures, and money and evaluate the performance.

The text of the thesis is structured as follows: In Chapter 1, we will dive into text normalization. Chapter 2 introduces all the theory used during implementation and evaluation. In Chapter 3, we present our design and implementation for the Czech text normalization task. Chapter 4 describes the data used for development and evaluation. Finally, Chapter 5 concludes the thesis with a summary of the results obtained and comparing the system with the Google TTS for Czech.

[1] https://themama.ai/

1. Text normalization

In this chapter, we present the task of text normalization (Section 1.1) and introduce semiotic classes of expressions that frequently require normalization (Section 1.2). We also discuss the general challenges of text normalization and show difficulties that may arise specifically in Czech or other morphologically rich languages (Section 1.3).

1.1 Task of the text normalization

From the point of view of TTS, an ideal text would consist only of fully spelled out words or names, and these spellings would be unambiguous, so that it would be clear from the written text which exact word was intended. Following Sproat et al. (2001), we name these fully spelled out expressions standard words. Unfortunately, texts often contain so-called non-standard words (NSWs) in addition to common words and names. These include numbers, abbreviations, acronyms, and all ideographic[1] words. This introduces ambiguity when converting the text into a spoken form: Take `123`, for example; it should be read as `one hundred twenty-three` in `123 pages`, but as `one twenty-three` in `123 King Ave`. Other examples of NSWs are presented in Figure 1.1.

Replacing NSWs with the contextually appropriate standard word or sequence of words is the task of text normalization. This process is necessary for automatic speech recognition (ASR) and text-to-speech (TTS) systems (Sproat et al., 2001). Based on the intended usage, we can further divide the text normalization task into social media and TTS domains.

1.1.1 Social media

A category that has received much attention in recent years is the normalization of social media texts – Hassan and Menezes (2013), Clark and Araki (2011), and Baldwin and Li (2015).

The main concerns of this domain are unintentional typographical errors and intentional non-canonical language, such as word-lengthening by duplication of characters. As this is not of interest to this thesis, we present only a few examples[2] to show the main idea, without discussing the topic deeper:

- abbreviation ("iirc" for "if I remember correctly"),
- internet slang ("that was well mint" for "that was very good"),
- phonetic substitutions and creative use of language ("gr8" for "great"),
- disguised vulgarities ("f***"),

[1] Representing an idea or concept directly, independent of any particular language, and specific words or phrases. For instance, `20 kg` is an ideograph representing the concept of 20 kilograms. The pronunciation may differ between languages – "twenty kilograms" in English, but "dvacet kilogramů" in Czech. That is the difference from `etc.` representing the exact phrase "Et cetera" albeit in an abbreviated form.

[2] The examples are inspired by Clark and Araki (2011).

- wordplay (“slooooow” for “slow”),
- emoticons (“:)” for a smiling face emoji),
- omitted punctuation (“Im” for “I’m”).

We refer the reader to the works cited above for a more detailed explanation.

1.1.2 Text-to-speech

One of the main differences between the TTS domain and social media text normalization is that some expressions, including most numerals, may be left unnormalized in information extraction from social media. However, normalization of such words is essential for TTS systems. On the other hand, the TTS normalization does not have to address all the phenomena from social media, as they can be directly pronounced.

Examples of sentences containing NSWs that need to be handled by TTS are shown in Figure 1.1. The first sentence is quite unambiguous. *Czech crowns* and *billions* are expanded, even though these may also be written in an abbreviated form. This is typical for newswire texts, where the clarity of the text is of concern. The second sentence is taken from Wikipedia, where number words and formulae are more frequent. Moreover, the precision is more important than the readability, and therefore the text is more prone to containing grammatically non-standard variants. According to Czech standard orthography, there should be no space between 40 and %. The last example is from an advertisement, where NSWs tend to be frequently used.

(1) Ministerstvo očekává, že takto vybere **25** miliard korun.
Ministry expects that this way it-collects **25** billion crowns.
‘The ministry expects to collect **25** billion Czech crowns in this way.’

(2) Reaguje s **40␣% H2SO4**
Reacts with **40␣% H2SO4**
‘Reacts with **40% H2SO4**’

(3) Kup **cca 500 g** za **125 Kč**
Buy **cca 500 g** for **125 Kč**
‘Buy **ca. 500g** for **125 CZK**’

Figure 1.1: Examples containing NSWs (in bold) from various domains.

A TTS text normalization system should cope with all of these cases and provide the correct spoken form. Before discussing the challenges, we will define the semiotic class.

1.2 Semiotic classes of non-standard words

For text normalization, it is helpful to divide the possible cases into groups with similar characteristics. In our text, we will use the term semiotic class to refer

to a categorical division of non-standard words (NSWs) defined by Taylor (2009, p. 93–95). Two taxonomies were introduced by Sproat et al. (2001) and van Esch and Sproat (2017). The latter extends the former and updates it to account for the changing form of texts on the internet.

The taxonomy we consider for domain-general normalization is shown in Table 1.1. In addition to selected classes taken over from van Esch and Sproat (2017), we include a new class labeled *with suffix*, i.e., expressions combining a numeral with a suffix, expressing a unit or substance. An example is `15krát` meaning "15 times". We added this class as we found it appropriate to have a separate class for this phenomenon, occurring in Czech and most other Slavic languages. The *percentage*, having an extra class in the original taxonomy is considered as part of the *measure* class.

Tokens are divided into three main categories: alphabetical, numeric, and miscellaneous. Within each category, tokens are subdivided depending on their verbalization and functional considerations of how the token is used. Note that some classes are not represented directly but are part of another class – e.g., Roman numerals belong to *cardinal* or *ordinal* numbers, depending on the context. Classes implemented by our solution in Chapter 3 are indicated in bold.

1.3 Challenges

We are facing several challenges when trying to normalize the text and number formats in particular. Firstly, we discuss the problems of accuracy, we continue with unrecoverable errors, and insufficient data, which are present in almost all languages. We conclude with morphological agreement, which arises as a problem in morphologically rich languages, including Czech.

1.3.1 Accuracy

An essential question for each system is the threshold from which the system is usable in the praxis. In applications such as automatic generation of video subtitles, we can also deploy the system when occasional errors occur. However, even low error rates cannot be tolerated in the case of navigational systems of a self-driving vehicle.

TTS systems are often among those where even rare errors are not tolerated. Moreover, the user expectations tend to be high. As mentioned by Ebden and Sproat (2015):

> "...if the system gets it wrong, listeners will immediately notice. In that case, it does not matter how good the voice quality is: at best, the system will sound like a stupid reader who happened to have a pleasant-sounding voice"

1.3.2 Unrecoverable errors

Unrecoverable errors are defined as linguistically coherent but not semantics-preserving (Zhang et al., 2019). An example may be normalizing `200 €` as "two

Category	Class	Examples
alphabetical	abbreviation	etc.
	letter sequence	OSN, EU
	read as word	NATO
	misspelling	geogaphy
	verbatim	#, *, =
numeric	**cardinal**	12, +45, -12
	ordinal	1., 123.
	decimal	13,15 or 3,1415
	date	12.12.2021, 7/4/2000
	time	3:20, 19.45
	measure	200 km, 40 Hz, 15 %
	money	1000 CZK, 0 Kč
	telephone	+420604586567
	identifier	042
	mixed	x220, 1080i50
	street address	Nerudova 42
	score	4:5, 45:15
	with suffix	24x, 20krát (20 times)
	chemical formulae	H2SO4
	mathematical expressions	4×6^{-2}
	fractions and ratios	3:2, 4/5
miscellaneous	funny spelling	slooow
	should be ignored	formatting symbols
	URL, Pathname, Email	www.mff.cuni.cz

Table 1.1: Non-standard words divided into semiotic classes based on van Esch and Sproat (2017) and updated by us for Czech language. Bold text denotes semiotic classes addressed by our toolkit, as described in Section 3.3.

hundred pounds". This type of error is not specific to the text normalization; for instance, we can also find it in the machine translation task.[3]

Consider the following situations. A TTS system reads `Take the 3rd` exit as "Take the three exit", the user will undoubtedly notice that the response is grammatically incorrect and therefore sounds unnatural, but will probably understand the meaning. Nevertheless, if the system says "Take the fourth exit", the user is misinformed and has no way of finding out.

This is also a problem when using accuracy as an evaluation metric (Section 2.2) because it simply cannot capture that not all mistakes are equally forgivable, and therefore a thorough error analysis is needed.

[3]Also named catastrophic errors. Consult Specia et al. (2020) for the current approaches on how these errors are mitigated in machine translation.

1.3.3 Insufficient amount of data

Today, most linguistic tasks are solved using machine learning models. This approach has the advantage of learning from data and removing the manual labor of crafting rules for these tasks. Nevertheless, at the same time, obtaining training data may be a problem. For state-of-the-art models such as BERT (Devlin et al., 2018) or T5 (Xue et al., 2020), large datasets are usually needed.

Moreover, there is a fundamental difference in collecting data for text normalization from, for example, machine translation, which is trained using parallel corpora. The emergence of translations included in parallel training corpora for machine translation is natural. On the other hand, datasets for text normalization must be specifically created or collected.

For social media text normalization (see Section 1.1.1), several training corpora exist – Clark and Araki (2011), Pennell and Liu (2011), and Yang and Eisenstein (2013). These include alternative spellings of words, such as `sloooow`, but do not include numbers that need to be transformed into spoken form.

For TTS normalization, English and Russian datasets were created for the Google Text Normalization Challenge[4] presented in Sproat and Jaitly (2016), based on Google's internal text normalization system. As the creation of datasets for the text normalization task is expensive, datasets are usually not publicly available for other languages.

1.3.4 Morphological agreement

As NSWs are abbreviated in nature, they typically omit any morphological inflection. The inflection, which is crucial for TTS, cannot be extracted easily and must be inferred from context. This is especially true for ideograms. We can observe this in English to some extent, where `1 kg` should be pronounced as "one kilogram", and with higher integers, the plural form "kilograms" should be used. However, the difficulty is most apparent in languages with a rich nominal inflection system, such as Czech, Polish or Russian. As opposed to English, where `2` is always spoken as "two", in Czech, multiple valid spoken forms exist based on the context (see Table 1.2).

Sentence in Czech	Normalized sentence	English translation	Normalized translation
Vidím 2 psy.	Vidím **dva** psy.	I see 2 dogs.	I see **two** dogs.
Vidím 2 kočky.	Vidím **dvě** kočky.	I see 2 cats.	I see **two** cats.
S 2 psy.	S **dvěma** psy.	With 2 dogs.	With **two** dogs.
Bez 2 dětí.	S **dvou** dětí.	Without 2 children.	Without **two** children.

Table 1.2: The difference in morphological complexity arising in text normalization for Czech and English.

[4] https://www.kaggle.com/datasets/google-nlu/text-normalization

2. Theoretical background

This chapter introduces essential theoretical concepts used in the implementation and evaluation of our normalization module. We begin with a discussion of existing systems and approaches for text normalization (Section 2.1). We continue with a brief description of the evaluation metrics used (Section 2.2). Finally, we present two concepts used in our implementation (see Chapter 3): finite-state machines (Section 2.3) and language models (LMs) (Section 2.4).

2.1 Related work

Text normalization has been part of TTS systems since the first deployments. Initially, hand-made rules were used. These developed into systems combining rules with a language model. Our implementation also falls into this category. The latest research focuses on neural networks.

2.1.1 Non-neural approaches

The first systematic research was presented by Sproat (1997), where the whole process of TTS-synthesis is solved using the weighted finite-state transducer (WFST) (Section 2.3). The architecture consists of two parts; *Lexical analysis phase*, which converts input text into a sequence of words with possible annotations, and *Grapheme to phoneme phase*, transforming words into phones. The Lexical analysis outputs all possible lexical categories for each word. To remove contextually inappropriate variants, language model transducers were used. These were derived from rules and other linguistic descriptions that apply to contexts wider than the lexical word. Newly introduced was the usage of WFST for digit and abbreviation expansion during the lexical analysis (in addition to WFST application to morphology, phonology and syntax).

The Kestrel system (Ebden and Sproat, 2015) focused purely on text normalization instead of the whole text-to-speech synthesis task. There are two main differences from the previous systems. The first is the usage of a morphosyntactic tagger instead of language model transducers. Still all possible forms marked with their lexical categories are generated using a WFST and only at the end, the correct variant is selected using the tagger. The second difference is splitting the normalization into two stages. The first stage *TokenizeAndClassify* transforms the input text into tokens, where each token is classified into one of the semiotic classes (see Section 1.2). Based on the assigned class, the second stage *Verbalize* produces the normalized form. Part of the Kestrel system was released under the name Sparrowhawk.[1]

In 2019 a normalization system for Polish[2] was published by Poswiata and Perelkiewicz (2019). The architecture takes advantage of the existing Polish morphological tagger (Wróbel, 2017), which is able to determine the morphosyntactic categories for numeric tokens. But in contrast to previous use of tagger in Kestrel

[1] https://github.com/google/sparrowhawk

[2] Note that Polish and Czech are pretty similar concerning complexity. Furthermore, there are no large dedicated datasets available for either.

(Ebden and Sproat, 2015), morphological tagging is performed as a second step after text tokenization. This allows the following components to generate just a single normalization variant. Training the tagger on the National Corpus of Polish (Przepiórkowski, 2012) data was possible because morphological categories are annotated for numeric tokens. This is different in Czech corpora, which typically use the Czech positional morphological annotation (Hajic, 2004). This includes the Prague Dependency Treebank (Hajic et al., 2006) and the Czech National Corpus,[3] where morphological categories for numeric tokens are omitted.

A recent approach proposed by Tyagi et al. (2021) builds on granular tokenization and two types of semiotic classes: (1) manually created classes capture the most frequent semiotic classes, and (2) automatic classes are learned from data. The tokenization starts by splitting at whitespace and then further on changes in the unicode class. Therefore, complex tokens such as dates are split into multiple consecutive tokens.

Each class defines which tokens it accepts and one particular way of translation for the accepted tokens. The whole task of text normalization is then regarded as a sentence tagging problem. There may be multiple matching classes for a single token. On that end, the authors stated:

> "In such cases of multiple matching classes, we pick the least frequent class to increase the representation of infrequent classes. This compensates for the imbalance present in the proportion of classes in the training set."

Whether this approach yields good results is not discussed in the paper and is left for further research. Note that for morphologically rich languages, there would be a separate class for each set of morphological properties.

2.1.2 Neural approaches

Since the year 2016, in which the Text Normalization Challenge competition was held, the task attracted more attention and early attempts to utilize the neural networks for this task were suggested (Sproat and Jaitly, 2016). The competition was separated into two subcompetitions; one for English text[4] as a representative of a morphologically simple language and one for Russian[5] as an example of a language with complex morphology.

Two different architectures were presented by Sproat and Jaitly (2016). The first consist of two long short-term memory (LSTM) based models: The first a bidirectional sequence-to-sequence model, called the channel, generates the possible normalizations and their probabilities, and the second, called the language model, chooses the correct normalization in the given context. During decoding, any channel model output with a high probability (more than 98%) is directly chosen. Also, outputs with a probability of less than 5% are pruned. Finally, only the first n options are left if the channel model outputs multiple variants. The language model is then used for selecting the correct variant in the context.

[3] https://www.korpus.cz/

[4] https://www.kaggle.com/c/text-normalization-challenge-english-language

[5] https://www.kaggle.com/c/text-normalization-challenge-russian-language

The second architecture is attention-based (Mnih et al., 2014) sequence-to-sequence model. Each token is placed in a window with three words to the left and three to the right, and the to-be-normalized tokens are marked in the input text.

Finite-state filters, i.e., rule-based systems generating possible verbalizations, are presented as an option to prevent the models from producing unrecoverable errors (Section 1.3.2).

The normalization task might seem like a simpler version of machine translation, where most words remain unmodified. Unfortunately, this is not true, as mentioned by Zhang et al. (2019). Applying state-of-the-art architectures for machine translation does not produce satisfactory results in the text normalization task, and specialized architectures are needed.

Zhang et al. (2019) thus introduce a specialized architecture for normalization based on gated recurrent units (GRUs). The targeted problem is that both the input and output are whole sentences,[6] which makes it much more challenging to correct the errors since it is hard to reconstruct, which output token(s) correspond to which input token(s). This motivates the authors to treat each token separately. The idea is to encode the left and right context into vectors using gated recurrent unit (GRU) (Cho et al., 2014) and then generate the normalized text for the token from the token itself and the context vectors using a bidirectional encoder GRU.

The described architectures based on deep recurrent neural networks (RNNs) achieve excellent results. Their significant advantage is the ability to learn from data and the possibility of using the same architecture for multiple languages.

However, using other solely machine-based solutions in real applications is complicated by two problems in particular, the lack of annotated data and the unrecoverable errors, as discussed in Section 1.3. Unrecoverable errors prevent direct use of neural models for text normalization, even in languages with sufficient data.

To avoid unrecoverable errors, covering grammars were introduced by Gorman and Sproat (2016) and Ng et al. (2017). A neural model generates a normalization of a NSW, and a hand-written grammar runs in parallel. The output of the grammar limits the possible outputs from the neural model, thus guaranteeing semantic accuracy. Moreover, the covering grammar is fundamentally language-universal, i.e., the developer only provides a lexicon of verbalizations for individual terms (e.g., how `PM` is spelled out). On the other hand, covering grammars produce a large number of disfluent variants and are thus unsuitable for direct use for the task of text normalization.

2.2 Evaluation metrics

Multiple evaluation metrics have been adopted for the text normalization task (Gorman and Sproat, 2016; Poswiata and Perelkiewicz, 2019; Tyagi et al., 2021). The sentence error rate (SER) and word error rate (WER) are the most prominent ones. Both are measured against reference sentences. The sentence error rate is

[6] Such tasks and corresponding architectures are called sequence-to-sequence problems and architectures, respectively.

defined as:

$$\text{SER} = \frac{\text{Wrong sentences}}{\text{Total number of sentences}}.$$

If the expected and evaluated sentences differ, even in a single token, the sentence is classified as wrong. The advantage of this metric is that it is easy to measure if reference texts are available. On the other hand, in most cases, we cannot get the accuracy per semiotic class as sentences may contain NSWs from multiple classes; therefore, this metric is not ideal for error discussion. A sentence accuracy as a complementary value to SER is also frequently used.

Another common option is the word error rate, derived from the Levenshtein distance and can be computed as:

$$\text{WER} = \frac{\text{Substitutions} + \text{Deletions} + \text{Insertions}}{\text{Number of Words}}.$$

In this metric, we must realize that most words go through unnormalized in a standard text. To accommodate this, we can use the number of words needing normalization instead of the total number of words in the denominator.

The third option is computing accuracy per semiotic class. For this, token-level semiotic class annotations are, of course, needed. Results provide us with more information: We know which classes are good enough and which should be worked on.

2.3 Finite-state machines

In this section, the finite-state machines used in our implementation are introduced in the form of examples. We start from a basic finite-state machine and generalize the concept up to a weighted finite-state transducer (WFST). We recommend Gorman and Sproat (2021) for a more detailed description.

2.3.1 State machines

A state machine is an abstraction described solely in terms of a set of states and oriented arcs, which represent transitions between those states. Each machine must have an initial state and may or may not have a final state. The state can be viewed as a memory of the machine, and the arcs as the operations.

Figure 2.1 shows an example of a state machine with two states (*Locked*, *Unlocked*), and three arcs. Such an finite-state machine represents a turnstile. The machine is initially in the *Locked* state. If the turnstile is pushed in this state, nothing happens. When a coin is inserted, the state changes to the *Unlocked* state, now if we push, we can pass it, and then the turnstile locks again.

2.3.2 Weighted finite-state acceptor

We now extend the finite-state machine idea to get a finite-state acceptor (FSA) and then further to obtain weighted finite-state acceptor (WFSA). The aim of a finite-state acceptor is to determine whether an input text corresponds to a particular pattern. The same can be achieved using a regular expressions. The

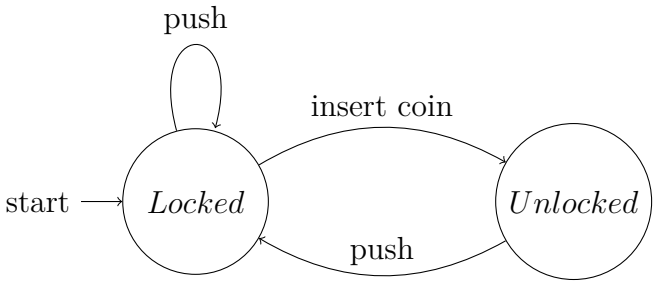


Figure 2.1: Example of a turnstile finite-state machine. The initial state is denoted with an arrow labeled "start" and there is no final state. The arcs are labeled with action descriptions.

FSA accepts the input, i.e., the input fulfills the pattern, if there is a path starting in the initial state and reaching the final state while reading the input string.

WFSAs allow the addition of weights to individual transitions and can be viewed as state machines where every arc is associated with an operation and a weight. Our goal is to find the shortest path or all paths from the initial to the final state. Path length is typically defined in terms of edge weights, such as a multiple of all edge weights along a path.

Instead of rigorously defining the WFSA (see details in Gorman and Sproat (2021)), we present an example in Figure 2.2.

The example WFSA accepts time fomat strings in the shape $hh : mm$. To demonstrate the WFSA's operation, let us assume that we want to check whether the string `02:11` satisfies the pattern. We start in the initial state S, then read the first symbol of our text `0`. From the text over the edges, we can see that `0` takes us to the state h. We continue reading the second symbol `2`. The edge from h accepts all digits; therefore, we can take it and are in state hh. Now we read the colon and get to the state : . Then we continue in the same style. After we read all the characters from the input text, we end up in the final state mm. Therefore, our automaton accepts the text.

If we try reading symbols not present on any outgoing edges, we are not accepting the text. An example of non-accepted inputs are `01:2a`, `34:11` or `en`.

Note that the weights in the example WFSA are trivial (all equal to 1). The use case for them is in case multiple paths exist from the initial to the final state. The weights are used to assign a score to each path through the automaton. The final score of a path is a function of the weights of the edges taken. If we take the sum as a function, the score of the above-mentioned example `02:11` would be 5.

2.3.3 Weighted finite-state transducer

The weighted finite-state acceptor can be further generalized to a weighted finite-state transducer (WFST). The main difference is that in a transducer, we can rewrite the text using rewrite rules that are tied to the edge transitions. We apply all the rules encountered during the best pass through the automaton. The result of the operation is then not only accepted or refused, but we get the output text capturing all the updates made by the individual edges.

In the example in Figure 2.3, we introduce a transducer for rewriting a number consisting of three digits to a text where single digits are transformed into their

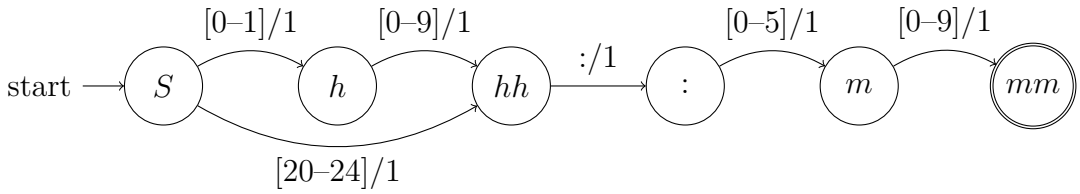


Figure 2.2: Example of a weighted finite-state acceptor for accepting time fomat strings in the shape $hh : mm$. The final state is, by convention, denoted by a double-struck line. Arcs labels consists of symbols and weights separated by a slash.

spoken form. To simplify the figure, we expect the digits to be just 1,2,3.

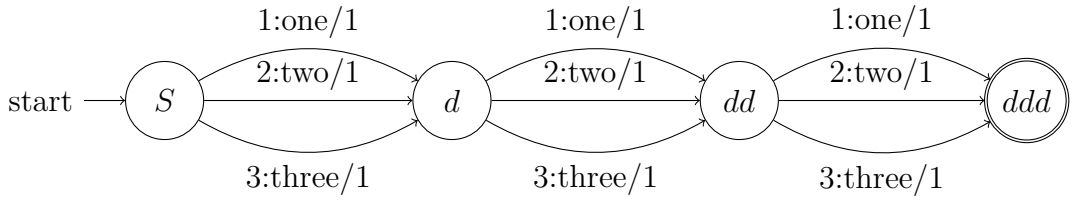


Figure 2.3: Example of a weighted finite-state transducer. Rewrite rules are shown in the form "input:output/weight". For instance, `123` gets rewritten into "one two three".

2.4 Language models

Language models are statistical models that assign probabilities to sequences of words. The first language models were based on n-gram models (Shannon, 1948), which given a context of $N-1$ words, predict the most probable word that follows. These were followed by the architectures based on RNNs (Mikolov et al., 2010). RNNs (Hochreiter and Schmidhuber, 1997) are designed to take sequences of text as inputs or return sequences of text as outputs, or both. The name recurrent comes from the fact that the output and state from each time step is used as input for the next one.

The RNN architectures were improved using the attention mechanism proposed by Bahdanau et al. (2015). This mechanism allows the model to choose how much "attention" to give to each input, depending on the current computation step. Given the success of the attention mechanism, new architecture called Transformer was introduced (Vaswani et al., 2017). The Transformer uses attention and fully connected layers to process input sequences. The main advantage of Transformers over RNN with attention is that they can be trained in parallel and are therefore faster to train.

The Transformer architecture is the basis of most of today's so-called pretrained language models – BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), GPT-2 (Radford et al., 2019), T5 (Raffel et al., 2019) and more. These are trained on vast datasets in a self-supervised fashion (i.e., no annotated data was used for the training, only raw texts). During this process, the language

model learns an internal representation of the language and can be further fine-tuned to a specific task, but this time on a significantly smaller dataset.

These models are further divided based on the mode of training. The *Masked models* are trained by guessing masked (removed) words from the sentence.

On the other hand, for *Causal models* such as GPT-2 (Radford et al., 2019), the objective is different. Given a sequence of tokens, their task is to predict the next token. Therefore only the left context is available as opposed to masked models, where the sentence context can be used to predict the masked word.

It is important to mention, given our use of the language model in Chapter 3, that causal models are more straightforward for computing the probability of a sentence.

3. Implementation

In this chapter, we present the architecture of our system, enclosed as Attachment A.1, along with the decisions that motivated each design approach. We start with a high-level architecture overview (Section 3.1). Then we present the selected programming environment (Section 3.2), list of and changes made compared to the theoretical division of semiotic classes (Section 3.3). We continue with a detailed description of the individual components of the system (Section 3.4–Section 3.8). At the end, the development dataset is briefly introduced (Section 3.9)

3.1 System Architecture

The current version contains four components: preprocessor, tokenizer & classifier, verbalizer, and ranking module. The architecture diagram is shown in Figure 3.1.

1. The input text is first processed by the preprocessor (Section 3.4), the purpose of which is to standardize the text (e.g., quotation marks) and clear text from formatting symbols (e.g., end of line). We refer to the preprocessor result as *standardized text* because we need to distinguish it from the result of the entire pipeline, which we call *normalized text*.

2. The *standardized text* is passed for tokenization (Section 3.5) and classification (Section 3.6), which are done in a single step. The text is divided into tokens suitable for further processing, and each token is assigned a semiotic class.

3. Tokens are further processed by a verbalizer (Section 3.7) that assigns possible pronunciations (verbalizations) to each token requiring normalization depending on its class.

4. The last step is the selection of the context-appropriate variant from the generated options. The ranking module (Section 3.8) scores each option; the lower the score the better the option.

Although none of the individual steps used are new, we are not aware of any system using the same architecture as ours. From the Kestrel design (Ebden and Sproat, 2015), we adopted the combination of tokenizer and classifier into one step and separation of this step from the subsequent verbalization. The use of finite-state machines (Section 2.3) in the implementation of these steps has also been used in the past. We chose to generate possible spoken forms using rule-based finite-state automata over neural networks (NNs) due to unrecoverable errors and lack of training data (both described in Section 1.3). Compared to the Kestrel architecture, we add an explicit preprocessing component that performs symbol standardization. Furthermore, since we do not have a suitable morphological

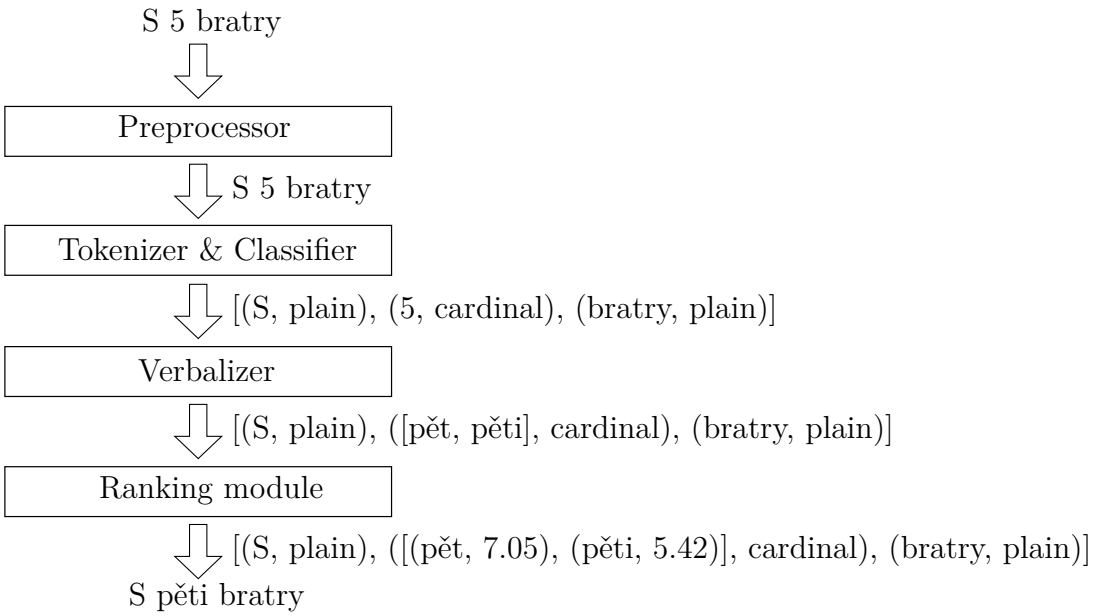


Figure 3.1: A high-level overview of the architecture with a Czech example of the data generated by the individual components. The same sentence is used as a running example when describing the individual components in the following sections. The sentence means "With five brothers", where "With" corresponds to "S", "five" to "pět" or "pěti" and "brothers" to "bratry".

tagger[1] available for Czech, we use a neural language model (Section 2.4), with the GPT-2 architecture, to rank the individual variants. The architecture also allows running without the ranking step and can generate output with fixed morphological properties for each class.

As we were trying to make the individual components independent, it was a natural solution to have stateless components and all normalization data extracted into a data class that the individual components modify.

The state of the normalization is kept in *NormalizerData*, containing the original text, standardized text – a result of preprocessor – and tokens. Each token consists of the original word, the semiotic class (Section 1.2) it belongs to, a span in the standardized text, and verbalizations with their scores. Note that the normalized text is not explicitly part of the class, as this can easily be obtained by replacing the tokens with their verbalizations in the standardized sentence.

3.2 Programming Environment

The first task was to choose a suitable programming language. We came up with two choices from our initial research: Python and C++. We have chosen Python because of its easy connectivity with machine learning and computational linguistics libraries. Furthermore, the WFSTs can be defined and compiled in Python

[1]Morphological tagger is a tool that assigns morphological tags to each word in a text, indicating its grammatical properties such as tense, grammatical case, number, and gender (Abney, 1997).

Our implementation	Original division
Cardinal	cardinal
Ordinal	ordinal
Decimal	decimal
Date	date
Time	time
Measure	measure, percentage
Money	money
Roman	cardinal or ordinal
Score	score
Telephone	telephone
With suffix	with suffix
Identifier	identifier
Mixed	mixed

Table 3.1: Overview of the implemented semiotic classes and the reference to the theoretical division presented in Table 1.1.

thanks to the Pynini[2] library, which is the most advanced library to work with WFSTs, based on the comparison presented by Gorman (2016). For the language model, we used PyTorch[3] in combination with HuggingFace Transformers.[4]

3.3 Changes in semiotic classes

During the actual implementation, we made several changes from the original theoretical division into semiotic classes (Table 1.1). An overview of the categories we implemented, including a reference to the original division, is given in Table 3.1. Firstly, the alphabetic and miscellaneous categories were not part of our focus, so we omitted them entirely. Also street addresses, chemical formulae, mathematical expressions, and fractions and rations were not included in our implementation. We merged percentages into the measure class because the use of both of these categories seems to behave identically in Czech.

On the other hand, we chose to have a separate category for Roman numerals. Although they are verbalized in the same way as cardinal or ordinal, they must first be correctly recognized in the input text, and therefore it is beneficial to study them as a separate class.

3.4 Preprocessor

The goal of the preprocessor is to prepare the text for subsequent processing, so that we save ourselves from dealing with special cases irrelevant to our task in later stages. The current version contains a few dozen rules we created to replace different types of quotes, spaces, and dashes with uniform variants and to remove non-printable characters and formatting symbols.

[2] https://www.openfst.org/twiki/bin/view/GRM/Pynini
[3] https://pypi.org/project/torch/
[4] https://pypi.org/project/transformers/

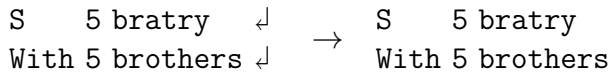


Figure 3.2: Preprocessing example, where the left sentence is transformed into the right one. The ↲ symbolizes a new line, which is stripped during the preprocessing. This and all the following examples in this chapter contain the original Czech text with a literal translation into English.

3.5 Tokenizer

The task of the tokenizer is to divide the input text into individual tokens, and our goal is to make the individual tokens form a meaningful parts for subsequent components. The usual choice is to use an existing tool, such as Natural Language Toolkit (NLTK)[5] or split the text on spaces and then further on the boundary of Unicode classes. The disadvantage of these options is the need to merge some tokens afterward, as numerical expressions tend to include different Unicode classes and need to be processed as a whole. An example is merging `1`,`/`,`1`,`/`,`2021` back into `1/1/2021` so that the token can be further processed as a date. Therefore, our implementation combines tokenization with the subsequent classification (see Section 3.6).

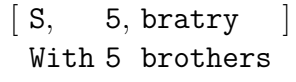


Figure 3.3: Tokenization example. For this and all the following examples in this chapter, the input is omitted as it corresponds to the output of the previous component.

3.6 Classifier

The classifier should determine each token's category (one of the semiotic classes for NSWs defined in Section 1.2 or *plain* if the token does not need a normalization).

We have implemented two classifiers. The first one is based on WFSTs (see details below in Section 3.6.1), whose main characteristic is assigning each token into just one category, and tokens cannot overlap.

The second one is based on regular expressions (Section 3.6.2), allowing overlapping tokens and thus assigning one piece of text into multiple categories. This classifier was developed primarily for experimental purposes.

We also considered using a named entity recognition (NER) system for Czech: Name Tag 2 (Straková et al., 2019). In this case, we would apply the NER system and get the NSWs recognized with entity types supported by Name Tag 2. The reason why we chose not to use this option is the effort to convert the recognized entity types into semiotic classes, which is not straightforward as the annotation schemes are not directly compatible.

[5] https://www.nltk.org/

Furthermore, not all NSWs are recognized by Name Tag 2 as named entities. This means that the remaining NSWs would then again have to be classified by one of the options mentioned earlier.

```
[ (S,    plain), (5, cardinal), (bratry,   plain) ]
  With          5              brothers
```

Figure 3.4: Classification output example. Each token corresponding to a semiotic class is given a corresponding label. If the token does not require normalization, then it is labeled *plain*.

3.6.1 WFST

The first and, at the present time, the only practically usable classifier is implemented using a WFST which tokenizes and classifies the input text simultaneously. The output of the finite-state automaton is an XML-tagged text.[6] The initial inspiration was taken from Sparrowhawk,[7] which is a public example of the Google's internal TTS Kestrel normalization system (Ebden and Sproat, 2015). The output tokens and corresponding labels are then extracted.

Both the tokenization and classification are performed by a single automaton, which is defined on about 500 lines of code. As described in Section 2.3, each possible path through the automaton is assigned a weight. The path with the lowest weight determines the division of the sentence into tokens and also determines the semiotic class of each token. All classes compete, and the weights are carefully set up to select the desired class, i.e., class that is expected to be the most frequent. If a part of text cannot be classified into any of the semiotic classes, then it is marked as *plain* (meaning no normalization is required).

3.6.2 Regular Expressions

Hand-crafted regular expressions form the basis of the second classifier. We created a general class called *General* with a single method *accepts*, which gets the input text and returns all matches found, already in the correct format required by the protocol.

For each semiotic class, an instance of the *General* class is initiated with the corresponding regular expression, matching all considered expressions. The *accepts* method is performed during the classification on all registered classes, and the union of matched tokens is returned.

This solution performs tokenization in a looser sense, where the non-classified text parts are considered a single token belonging to the *plain* class.

There are known shortcomings, for example, that not all digits are necessarily normalized due to the language model choosing a different variant. Although we publish the classifier to present another approach to the problem.

[6] The output format is solely our decision, as the WFSTs can perform arbitrary rewrites.
[7] https://github.com/google/sparrowhawk

3.7 Verbalizer

The purpose of the verbalizer is to generate all allowable variants for NSW tokens. Because the number of possible variants may be non-trivial, the verbalizer can be configured to generate only a subset of expansions based on their morphological properties. In the following, we will go through all supported semiotic classes and their individual verbalizer implementation in our solution. There is a separate Section 3.7.1 for the cardinal and ordinal numbers, as they are more complex. The rest classes are described together in Section 3.7.2. For a verbalization example of individual classes, see Figure 3.2.

```
[ (S,    plain), ([pět,      pěti      ], cardinal), (bratry,  plain) ]
  With       five.NOM five.INS                       brothers
```

Figure 3.5: Verbalization output example. We indicate in which grammatical case the corresponding numeral is inflected (NOM stands for Nominative, INS stands for Instrumental).

3.7.1 Cardinal and ordinal numbers

The difference between the cardinal and ordinal numbers from the other classes comes from the fact that the verbalization of other classes depends on it. The core of the verbalization are again WFSTs.

We use a split into two steps, factorization and verbalization, as described by Ebden and Sproat (2015).

In the factorization step, we transform the numbers from their raw form into a form where significant powers are explicitly marked, see Figure 3.6. Factorization is complicated by the Czech nominal system, where the word for hundred, thousand, etc. changes depending on multiple previous digits, e.g., 2000 is pronounced as "dva *tisíce*" (two thousand), but 22000 as "dvacet dva *tisíc*" (twenty-two thousand).

In the verbalization step, the factorization result is verbalized according to the specified gender and case for cardinal numbers. Ordinal numbers add extra grammatical category *number*.

10	→	1[E1]
1000000	→	1[E6]
1234	→	1[E3]2[E2]3[E1]4

Figure 3.6: Example of number factorization as perfomed by WFST. The [E6], [E3], [E2] and [E1] symbolize the places where the words for million, thousand, hundred, and tens, respectively, should be generated. Note that the following verbalization still depends on the context: 1[E1] is expanded to "deset" (ten), but 2[E1] to "dvacet" (twenty).

3.7.2 Verbalizations of individual classes

Decimal number category builds upon the cardinal numbers and offers two verbalization options. A long variant where a name for the fractional part (tenth, hundredth, thousandth) is explicitly pronounced – `4,3` is expanded as "čtyři celé tři desetiny" (four whole three tenths). The short variant omits the name of the fractional part, resulting in "čtyři celé tři" (four whole three).

Date expressions offer a single variant, but still multiple expansions based on the grammatical case. An arbitrary combination of day, month, and year can be provided.

Time also offers short and long variants, with the long variant explicitly mentioning hours, minutes, and seconds – pronouncing `4:25` as "čtyři hodiny a dvacet pět minut" (four hours and twenty-five minutes). And the short variant only reads the numbers contained in the expression – transforming the above example into "čtyři dvacet pět" (four twenty-five).

Measure class expressions verbalize both number and unit, thus ensuring grammatical congruency (in grammatical gender and case). The units are split into two categories: (1) units that can have a prefix (mili, kilo, tera, etc.), such as hertz (Hz), tesla (T), and gram (g), and (2) units that cannot, e.g., percent (%), hour (h) and parts per million (ppm). Furthermore, the single units can be combined into compound ones (e.g., km/h). For the compound units, we also distinguish situations where "za" (per) and "na" (per) are used. Examples of this are to be found in Figure 3.2. For the number verbalization, the rules for cardinal and decimal numbers are used.

Money expressions are handled similarly to measures. In this case, the token is split into number and currency. At the time of writing, we support the 25 most traded currencies,[8] including the Czech crown.

Score tokens are verbalized, either as two numbers, e.g., `3:3` as "tři tři" (three three) or by adding "ku" (to) between the numbers and inflect the second accordingly, e.g., "tři ku třem" (three to three).

Telephone numbers are verbalized digit-by-digit or by groups of three. If an area code is present, it is read as a single number.

With-suffix words are always generated as a cardinal digit in the nominative or the locative case concatenated with the suffix.

Identifier expressions are always verbalized with numbers spelled out digit-by-digit.

[8] https://en.wikipedia.org/wiki/Currency

Semiotic class	Text	Verbalization	Translation
cardinal	35	třicet pět	thirty-five
	-3	mínus tři	minus three
ordinal	321.	tří stý dvacátý první	three hundred and twenty-first
		tří stá dvacátá první	three hundred and twenty-first
decimal	14,2	čtrnáct celých dvě desetiny	fourteen whole two tenths
		čtrnáct celá dva	fourteen whole two
date	12. března	dvanáctého března	the twelfth of March
	3/12	třetího prosince	the third of December
time	3:20	tři hodiny dvacet minut	three hours twenty minutes
	3:20	tři dvacet	three twenty
measures	100 m	sto metrů	hundred meters
	5 m/s	pět metrů za vteřinu	five meters per second
	5 Pa/m	pět pascalů na metr	five pascals per meter
money	1000 CZK	tisíc korun českých	thousand Czech crowns
scores	4:5	čtyři ku pěti	four to five
		čtyři pět	four five
telephone	+420604586567	plus čtyři sta dvacet	plus four hundred and twenty
		šest set čtyři	six hundred and four
		pět set osmdesát šest	five hundred and eighty-six
		pět set šedesát sedm	five hundred and sixty-seven
	+420604586567	plus čtyři sta dvacet	plus four hundred and twenty
		šest nula čtyři pět osm	six zero four five eight
		šest pět šest sedm	six five six seven
with suffix	20x	dvacet krát	twenty times
	32bit	třiceti dvou bitový	thirty two bit
identifier	042	nula čtyřicet dva	zero forty-two
mixed	x220	x dvě stě dvacet	x two hundred and twenty
		x dva dva nula	x two two zero

Table 3.2: Examples of verbalizations of semiotic classes. The variations shown represent the possible phenomena described in Section 3.7.2.

Mixed tokens, such as `H12C8` are also verbalized in two variants, digit-by-digit – "H jedna dva C osm" (H one two C eight) – or numeric sections are read as cardinal numbers – "H dvanáct C osm" (H twelve C eight).

3.8 Ranking module

If the verbalizer generates multiple possible options, we must select the contextually correct one from these options. This selection is the task of the ranking module.

Our implementation supports two modes of operation, with and without a language model (LM). If LM is not set and no verbalization is specified, then there is a default setting that generates one variant for each class. In the variant without a LM, the system assumes that the user has specified a single verbalization to be generated for each class.[9]

The second mode uses the LM to score the verbalizations. The language model can be present locally or downloaded from the HuggingFace website. In both cases, it should be a causal language model, compatible with the HuggingFace

[9]If there is no LM and user allows multiple variants, then the first variant generated is selected - but this is not how the system should be used.

transformers library interface. We decided on three Czech models based on small version of GPT-2 architecture because of the availability of these models and the expectation of lower latency. The first is spital/gpt2-small-czech-cs[10] trained only on Czech Wikipedia and with a permissive license, also allowing commercial use. The second is lchaloupsky/czech-gpt2-oscar[11] trained on the Czech part of the OSCAR[12] dataset and also having an permissive license. The last model considered is MU-NLPC/CzeGPT-2.[13]

We use the LM to score the sentence perplexities (Manning and Schütze, 2001, Section 2.2.8). Specially, we use a variant of the calculation for fixed-length models as described by HuggingFace.[14] If a model assigns a lower perplexity to a sentence, it deems it to be more probable.

The situation is straightforward if the sentence contains only one token to normalize. We replace the token with its verbalization and compute the sentence perplexity. The resulting score is assigned to the verbalization used.

If multiple tokens requiring normalization are present, we tried three variants:

1. The first option is to only consider a single token in isolation and leave the others in unnormalized form, then the perplexity is calculated in the same way as in the case of a single token – we refer to this option as "individual ranking".

2. The second variant, called "left-to-right ranking", goes from the start of the sentence and finds the best verbalization for the first token found. When the second token is evaluated, the already chosen verbalization of the first token is inserted into the sentence.

3. The last and further not considered option was to replace all tokens with the default verbalization, then continue as in the second option. In the course of testing, this method proved to confuse the language model.

After assigning a score to each verbalization, the normalized sentence can be built by selecting the verbalization with the lowest score (perplexity) for each token.

[(S,	plain), ([(pět, 7.05), (pěti, 5.42)],	cardinal), (bratry,	plain)]
With	five.NOM five.INS		brothers

Figure 3.7: An example of the output of a ranking module where, in the case of multiple generated pronunciations, each of them is assigned a score. The lower the score, the more suitable the variant is. We indicate in which grammatical case the corresponding numeral is inflected (NOM stands for Nominative, INS stands for Instrumental).

[10] https://huggingface.co/spital/gpt2-small-czech-cs
[11] https://huggingface.co/lchaloupsky/czech-gpt2-oscar
[12] https://huggingface.co/datasets/oscar
[13] https://huggingface.co/MU-NLPC/CzeGPT-2
[14] https://huggingface.co/transformers/v4.10.1/perplexity.html

3.9 Development dataset

During the development process, a small dataset was gradually created on which we tested our system and analyzed the errors. The dataset contains mainly sentences from the Czech Wikipedia, but it also contains sentences from other sources (e.g., Seznam news,[15] iDnes[16]) or even sentences made-up or translated by the author. For the evaluation it was necessary to annotate this dataset as well. In this case, we chose a more time-efficient option than for the evaluation dataset (Section 4.4). We let the normalizer generate possible annotations and then the author reviewed and corrected them. In doing so, what was corrected were mainly genuinely wrong forms or unrecognized entities, since the annotator (author) was certainly influenced by the knowledge of the normalizer's capabilities. The resulting dataset contains 650 sentences.

Another important point was, that although we have implemented the ability to generate many variants for most classes, in the end, we tried to reduce the number of variants as much as possible. The reason is that increasing the number of variants including infrequent ones makes it harder for the LM to choose the correct one. The reduction proceeded as follows. First, we set the system to generate the most frequent variant for each semiotic class. After running in this setting on the development dataset, we went through the variants that were not reachable and tried to add them to the system. After an update in the setting, we checked if the accuracy increased and thus whether to include this change or not.

[15] https://www.seznamzpravy.cz/
[16] https://www.idnes.cz/

4. Data

Since we decided to obtain at least a minimum amount of human-annotated data, which will form the first public Czech number normalization dataset to the best of the author's knowledge, it was necessary to prepare the raw sentences and set up an annotation process. We used the newly created datasets to establish a baseline and to determine whether changes to the implementation yield better results.

We begin the chapter by discussing the possibility of harvesting data from the web (Section 4.1). In Section 4.2, the annotated data format is introduced. We continue by presenting the newly created data annotation application (Section 4.3) and the associated annotation process (Section 4.4). Finally, the obtained datasets are described in Section 4.5.

4.1 Harvesting from the web

Harvesting naturally occurring data from the web is a possible solution for collecting a sufficient amount of data for some machine learning tasks, such as machine translation. We also considered using web data harvesting to collect text normalization data. Such an approach involves finding the already expanded forms of numerals and then denormalizing them back into numbers.

Even though this approach works quite well for cardinals and ordinals (i.e., numerals fully spelled out as words) up to 100, coverage of other cases is far from ideal. The major problem arises from the fact that some expressions are almost always written in numerical form. While it is quite common to write `3,000,000` as `three million`, expressing `3,456,799` in the expanded form is very unusual. This affects multiple other semiotic classes, such as time, measure, and money expressions. Therefore, the most often normalized terms would be missing from the datasets.

4.2 Data format

To decide on the annotation format, we first checked formats already in use – specifically, the Polish normalization dataset[1] (Poswiata and Perelkiewicz, 2019) annotated by human annotators, and the Russian and English datasets created by Google's internal hand-written Kestrel normalization system (Sproat and Jaitly, 2016). While the Russian and English datasets do not fit our purpose,[2] the Polish structure fits without significant changes.

The dataset contains individual sentences, each containing an ID, the dataset name, the original and the normalized sentence, and a list of to-be-normalized tokens. Each such token consists of the unnormalized part of the text and its span

[1] https://github.com/rafalposwiata/text-normalization

[2] Their format requires splitting the text into tokens and classifying each into one of the classes or marking it as *plain* (no normalization needed). This task is easy for a program. In our case, it would mean either requiring human annotator to tokenize the text or suggesting a possible tokenization and letting the annotator modify it. In either case, this is a redundant task.

in the original text, the corresponding semiotic class, and nested tokens called children – making it possible to subdivide the to-be-normalized token (e.g., a date can be further divided into a day, month and year), see Figure 4.1.

4.3 Annotation application

Apart from the data itself, we also lacked an appropriate annotation tool for the task. Therefore, we decided to create a web application specially aimed at annotating text normalization data. Because the application does not directly depend on our normalizer implementation described in Chapter 3, we developed it as an entirely independent project. The interface can be easily adjusted to annotate data in any language and is freely available online.[3] The annotation process is shown in the Figure 4.2.

As part of the design, we created a manual for the human annotators, enclosed as Attachment A.2. There, the annotation environment navigation and functions are presented. Also, examples of the semiotic classes are shown along with possible verbalizations. We tried to capture all the valid verbalizations we could think of to avoid constraining the annotator. As the intentional users of the application must have knowledge of Czech, both the system and manual are in Czech.

4.4 Annotation workflow

We decided to obtain the raw data from the Czech Wikipedia[4] and Czech Radio[5] websites. Czech Radio represents arguably clean text with minimal normalization requirements. On the other hand, Wikipedia represents more challenging text requiring a lot of normalization and containing frequent non-standard word forms.

The WikiExtractor tool (Attardi, 2015) was used to extract data from the Czech Wikipedia.[6] The data was then processed using a preprocessing script,[7] splitting the text into sentences and only extracting sentences containing numerals. NSWs not containing digits were not specifically searched, they are part of the dataset due to their natural occurrence in sentences with numerals. Sentences for annotation were selected at random from the data extracted.[8] Data from the Czech Radio website was scraped and then searched for digits manually.

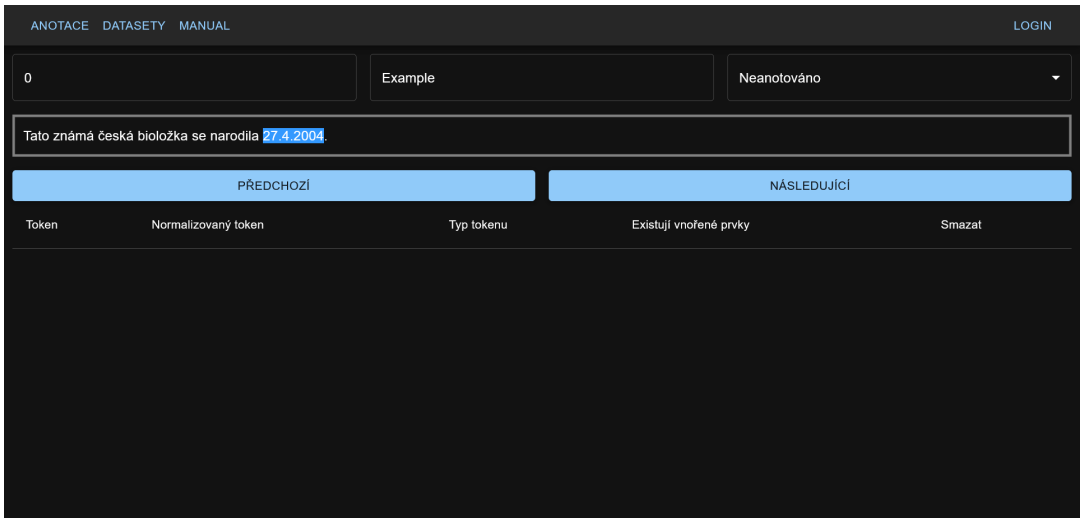
To obtain data for the evaluation, we deployed the annotator application on the author's web server[9] and asked family members to annotate (hundreds of sentences). The reason for delegating this task to others was to limit our knowledge about the test set.

A short individual training session was conducted with each annotator before the actual annotation started. This consisted of a mock annotation of 10–15

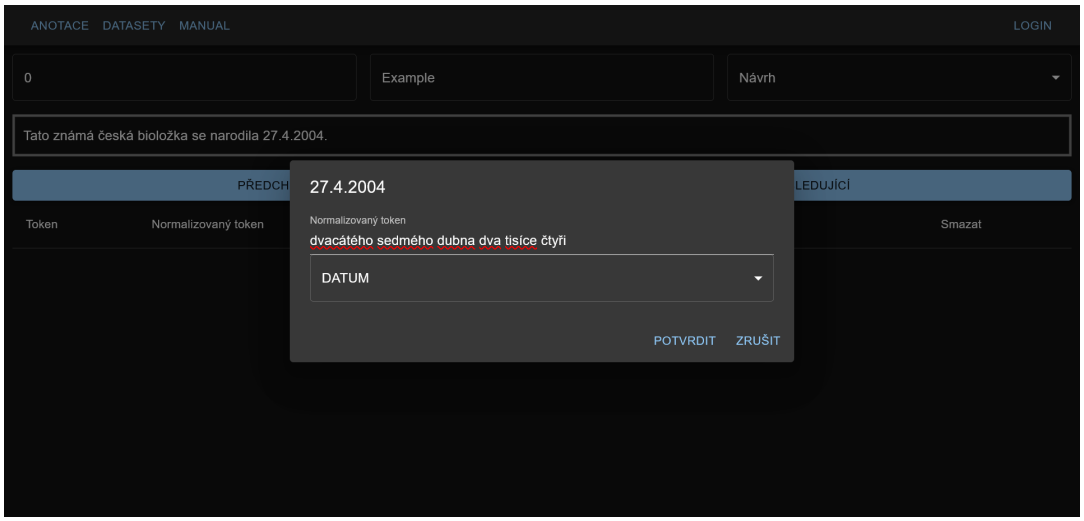
[3] https://github.com/RuzickaJakub/text-normalization-annotation-application
[4] https://cs.wikipedia.org/
[5] https://portal.rozhlas.cz/
[6] The data extraction was performed on 20.02.2022
[7] Attached under `tts_normalization/scripts/prepare_data.py`.
[8] Script is attached under `tts_normalization/scripts/choose_random_sentences.py`.
[9] A live annotation application is running on https://annotator.ruzickajakub.eu/.

```
{
   "dataset_name": "Example",
   "sentence_id": 0,
   "sentence_state": "Proposal",
   "original_sentence": "Tato známá česká biložka se narodila
                        27.4.2004.",
   "normalized_sentence": "Tato známá česká biložka se narodila
                           dvacátého sedmého dubna dva tisíce čtyři.",
   "tokens": [
      {
         "text": "27.4.2004",
         "normalized_text": "dvacátého sedmého dubna dva tisíce
                           čtyři",
         "type": "Date",
         "children": [
            {
               "text": "27.",
               "normalized_text": "dvacátého sedmého",
               "type": "Day",
               "children": [],
               "begin": 0,
               "end": 3
            },
            {
               "text": "4.",
               "normalized_text": "dubna",
               "type": "Month",
               "children": [],
               "begin": 3,
               "end": 5
            },
            {
               "text": "2004",
               "normalized_text": "dva tisíce čtyři",
               "type": "Year",
               "children": [],
               "begin": 5,
               "end": 9
            }
         ],
         "begin": 38,
         "end": 47
      }
   ]
}
```

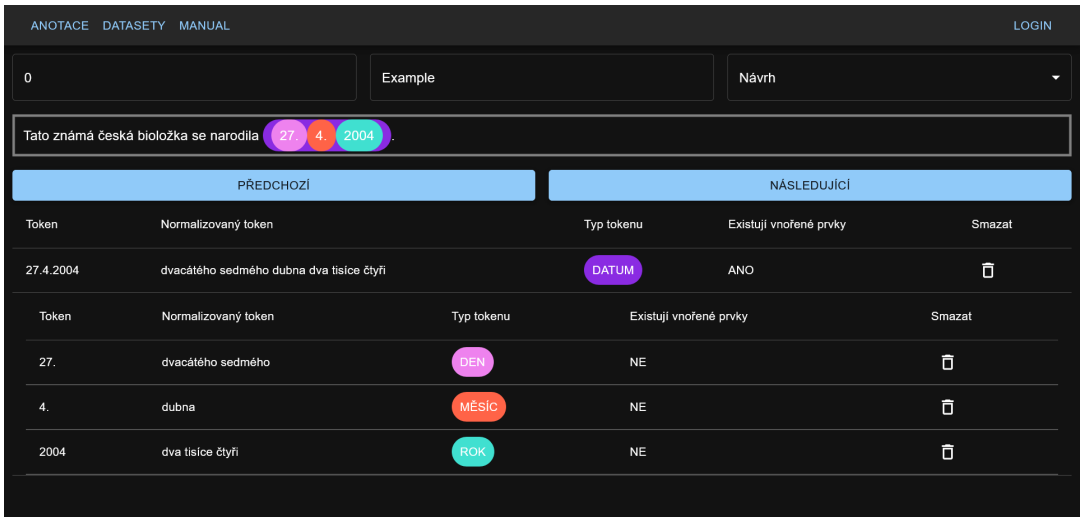

Figure 4.1: Example of an annotated sentence. The format is described in Section 4.2. The original sentence can be translated as: This famous Czech biologist was born on 27.4.2004.



(a) First, the part of text needing annotation is interactively selected.



(b) In the second step, the original text is shown to the user in the top left corner. The normalization of the text and corresponding semiotic class are filled in.



(c) When all the words to-be-normalized are marked, the annotator changes the sentence state from *Unannotated* to *Proposal*

Figure 4.2: A description of the annotation process.

sentences, where the annotator performed the annotation, and then the author provided feedback.

All sentences were double-checked and agreed by two independent annotators. In the first round, annotators were given a set of raw sentences and instructed to mark the words needing normalization, assign them to one of the semiotic classes, and write the contextually correct expanded form. In addition, the entire sentence has been set as *Proposal* and, in exceptional cases, as *Invalid* if the sentence is formed incorrectly. At this stage, the reported speed was between 20–40 sentences per hour. The tediousness of the process was due to the occasional need to find or think about the correct normalization.

In the second round, sentences marked as *Proposal* were selected from the annotated datasets, and each annotator was given sentences processed by someone else. An annotator who is already working with proposed sentences is called a second annotator. If the second annotator agrees with the proposal, the sentence is approved. If the second annotator disagrees with the proposed normalization, the sentence is left in the *Proposal* state and therefore not used in the dataset. Finally, if the second annotator agrees with the normalization, but it contains a typographical error, e.g., "červnci" instead of "červenci" (July), then the second annotator corrects the error and approves the sentence. The final datasets only include sentences approved during the second round.

4.5 Newly created datasets

The result of the above process are two small datasets for two domains – Czech Radio (News) and Wikipedia. The former contains 398 sentences and 599 to-be-normalized tokens, the latter 1,484 sentences and 2,586 to-be-normalized tokens. The numbers for each semiotic class are given in Table 4.1.

From the dataset generation process, it can be seen that the average representation of each class in a given domain corresponds to the actual occurrence rate among sentences with numerical expressions. Therefore, the accuracy on this data should be close to the actual accuracy on data from the same domain.

Furthermore, it can be noticed that on the Wikipedia dataset, the classes are more heterogeneous. In contrast, the Czech Radio dataset mainly comprises only a few classes. Phone numbers do not appear even once in the dataset. Similarly, some other classes (e.g., pathname, URL) are rare. A special dataset would be needed for use in a domain with a frequent occurence of these classes.

Semiotic class	Tag	Wikipedia	Czech Radio (News)
cardinal	Cardinal	1,220	367
ordinal	Ordinal	236	58
decimal	Decimal	32	39
date	Date	291	48
time	Time	17	15
measure	Measure	212	5
money	Money	16	5
roman	Roman	50	2
telephone	Telephone	0	0
score	Score	15	32
with suffix	WithSuffix	22	2
identifier	Identifier	22	7
mixed	Mixed	107	6
abbreviation	Abbreviation	157	5
chemical formulae	Chemistry	18	2
mathematical expressions	Math	6	0
pathname	Pathname	1	0
range	Range	115	3
URL	URL	2	3
verbatim	Verbatim	38	0
other	Other	9	0
Total to-be-normalized tokens	-	2,586	599
Total sentences	-	1,484	398

Table 4.1: Number of to-be-normalized tokens in our datasets.

5. Evaluation

In this chapter, we present our evaluation results, comparing a baseline, variant without language model and three variants with language models described in Chapter 3, using the data described in Chapter 4 for evaluation. Firstly, we describe the system variants (Section 5.1) and our evaluation methods (Section 5.2). We continue with a presentation of our results, detailed error analysis (Section 5.3 and Section 5.4) and comparison with Google TTS (Section 5.5). The chapter concludes with a discussion of possible improvements (Section 5.6).

5.1 Compared system variants

During the evaluation, we consider the following system variants:

Baseline. The first variant is based on the num2words library[1] for number to words expansion, which we improved to provide us with a decent baseline.[2]

Without LM. In this variant, only one expansion is generated for each to-be-normalized token.

Oracle. This option captures the idea that every time the correct verbalization is produced, it is also chosen. Therefore, this option represents an upper bound for our system. In some cases, not all variants the system is capable of are generated, because with the rising number of variants, the task of selecting the correct one becomes harder. This fact proves to be critical during the development.

With LM. As we do not have an oracle to magically choose the correct variant, we tried three neural models – lchaloupsky/czech-gpt2-oscar, MU-NLPC/CzeGPT-2, spital/gpt2-small-czech-cs – with two modes of scoring – individually and left-to-right. For details check Section 3.8.

5.2 Evaluation methodology

The module was evaluated on the Wikipedia and Czech Radio datasets described in Chapter 4 using the evaluation metrics described in Section 2.2. We computed both the sentence accuracy and accuracy for the individual semiotic classes.[3]

The sentence accuracy for the oracle is a very close approximation because sentences with a higher number of to-be-normalized tokens needing normalization can generate a vast number of variants due to combinatorial explosion. If the number of variants exceeds 100,000, we skip the sentence.[4]

[1] https://pypi.org/project/num2words/

[2] We created a pull request with our improvements to the original project (https://github.com/savoirfairelinux/num2words/pull/504).

[3] The evaluation script can be found in the attached files.

[4] In our case, the number of skipped sentences is minimal – 2 and 3 sentences on Czech Radio and Wikipedia dataset, respectively.

5.3 Results

On the Wikipedia dataset, our module achieves a 81.67% sentence accuracy in the best setting (model MU-NLPC/CzeGPT-2 with scoring variant left-to-right). The oracle variant achieves approximately 91.98% sentence accuracy. Table 5.1 shows the accuracy for individual classes, including the number of occurrences in the dataset.

Semiotic class	#tokens	baseline	without	oracle [%]	spital		oscar		MU-NLPC	
					indi [%]	left [%]	indi [%]	left [%]	indi [%]	left [%]
Sentence acc.	1209	11.99	72.79	91.98	69.31	70.31	77.42	78.74	80.56	81.47
Word accuracy										
Cardinal	1207	36.04	89.48	98.09	83.60	83.35	91.96	92.29	93.95	93.54
Ordinal	234	8.97	20.94	95.30	70.09	70.09	70.94	70.51	75.21	75.21
Decimal	31	6.45	58.06	80.65	29.03	35.48	64.52	67.74	64.52	70.97
Date	291	5.50	94.50	97.25	67.01	65.98	79.38	81.44	81.10	82.13
Time	16	68.75	81.25	81.25	62.50	62.50	68.75	75.00	75.00	68.75
Measure	209	0.00	60.29	77.51	49.28	49.76	59.33	59.33	65.07	64.59
Money	16	0.00	50.00	62.50	50.00	50.00	43.75	43.75	43.75	43.75
Roman	49	0.00	57.14	75.51	46.94	48.98	53.06	55.10	55.10	53.06
Score	15	93.33	93.33	93.33	93.33	93.33	93.33	93.33	93.33	93.33
With suffix	22	0.00	31.82	86.36	45.45	45.45	45.45	50.00	63.64	45.45
Identifier	22	4.55	36.36	36.36	36.36	36.36	31.82	31.82	36.36	36.36
Mixed	99	0.00	94.95	95.96	89.90	89.90	89.90	90.91	88.89	91.92
All	2211	22.61	77.79	93.76	74.27	74.17	82.09	82.72	84.67	84.49

Table 5.1: Our system evaluation on the Wikipedia dataset. To fit the space we use *indi* for individually and *left* for left-to-right scoring variant. Individual language model are referenced using the following abbreviations: spital (spital/gpt2-small-czech-cs), oscar (lchaloupsky/czech-gpt2-oscar), MU-NLPC (MU-NLPC/CzeGPT-2).

On the Czech Radio dataset, we achieved the best results with the same configuration as on the Wikipedia dataset (model MU-NLPC/CzeGPT-2 with scoring variant left-to-right). In this setting, our module achieves a 85.19% sentence accuracy. Furthermore, the correct sentence variant is generated for approximately 92.73% of sentences. The complete results are shown in Table 5.2.

Our results provide a hint for future implementations, showing for which classes it makes sense to generate a larger number of variants and for which categories this option can be omitted, in exchange for a possible slight reduction in accuracy. We can observe that for some of the classes, there is not a sufficient number of to-be-normalized tokens to draw conclusions, especially in the Czech Radio dataset.

The model comparison is clear-cut. The MU-NLPC/CzeGPT-2 outperforms both other models on both datasets in terms of sentence and word accuracy. At the same time, it achieves the best results in almost all semiotic classes (the only exception being dates on the Czech Radio dataset). The lchaloupsky/czech-gpt2-oscar performs only slightly worse than the MU-NLPC/CzeGPT-2 in terms of both sentence and word accuracy. The third evaluated model, spital/gpt2-small-czech-cs, achieves significantly lower accuracy, primarily due to poorer performance on cardinal and ordinal numbers. However, it is worth noting that the performance of the models can vary depending on the specific task and dataset.

In the following section, we discuss the results obtained using the best MU-NLPC/CzeGPT-2 setup in the left-to-right scoring variant.

Semiotic class	#tokens	baseline	without	oracle [%]	spital		oscar		MU-NLPC	
					indi [%]	left [%]	indi [%]	left [%]	indi [%]	left [%]
Sentence acc.	385	38.18	71.69	92.73	64.68	64.94	81.56	81.56	84.94	85.19
Word accuracy										
Cardinal	367	64.31	80.65	96.73	78.20	78.47	90.74	91.28	92.64	92.64
Ordinal	58	37.93	10.34	98.28	62.07	58.62	74.14	79.31	87.93	79.31
Decimal	39	2.56	66.67	71.79	41.03	46.15	41.03	41.03	56.41	58.97
Dates	48	8.33	95.83	100.00	58.33	58.33	95.83	95.83	89.58	91.67
Time	15	6.67	80.00	80.00	53.33	60.00	66.67	66.67	73.33	73.33
Measure	5	0.00	40.00	60.00	0.00	0.00	20.00	20.00	20.00	20.00
Money	5	0.00	80.00	80.00	80.00	80.00	80.00	80.00	80.00	80.00
Roman	2	0.00	0.00	100.00	50.00	50.00	50.00	50.00	50.00	50.00
Score	32	96.88	96.88	96.88	96.88	96.88	96.88	96.88	96.88	96.88
With suffix	2	0.00	0.00	100.00	50.00	50.00	50.00	50.00	50.00	50.00
Identifier	7	14.29	100.00	100.00	85.71	85.71	100.00	100.00	85.71	100.00
Mixed	6	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
All	586	50.51	74.40	94.71	72.35	72.70	85.15	86.01	88.23	87.88

Table 5.2: Our system evaluation on the Czech Radio dataset. To fit the space we use *indi* for individually and *left* for left-to-right scoring variant. Individual language model are referenced using the following abbreviations: spital (spital/gpt2-small-czech-cs), oscar (lchaloupsky/czech-gpt2-oscar), MU-NLPC (MU-NLPC/CzeGPT-2).

Cardinal and ordinal number: For cardinal and ordinal numbers, it certainly makes sense to use the language model, as it provides a non-trivial improvement for cardinal numbers (4% on Wikipedia and 12% on Czech Radio) and a crucial improvement in the case of ordinal numbers (54% on Wikipedia and 77% on Czech Radio).

Decimal number: The result is not clear-cut for decimal numbers – 6% gain on Wikipedia, but 10% loss on Czech Radio. Given the observed errors discussed in Section 5.4, it is still advisable to choose the language model version here.

Date and time: It is better not to give the model a choice for dates and times since, in most cases, a single version with particular morphosyntactical properties is correct. Adding more options, in this case, on the other hand, reduces the accuracy of the whole system – it results in 13% and 6% drop on dates and 6% and 7% in case of times on Wikipedia and Czech Radio, respectively.

Measure: The system scored lower on the measures class than we expected, based on development experience. The most significant shortcoming of our system is the lack of rules for recognition of less typical units, such as horsepower.

Money: For money, even the oracle (i.e., the correct variant being reachable) is already low. We observe a higher variety of pronunciation options than in other classes and a less standard notation for some currencies. Although it is not obvious from the scores, the use of the language model brings an improvement on the test data (see Section 5.4).

Roman numerals: On the roman numerals, our system achieves similar scores with and without the language model – 2% drop in accuracy when using the language model on Wikipedia, the Czech Radio dataset is not indicative, as it

contains only 2 roman numerals. We can observe, that by selecting a correct variant from the proposed verbalizations we can achieve up to 22% improvement.

With-suffix: For the with-suffix class, the language model brings a clear improvement. Since most words with a suffix are in the nominative or locative case, the language model only has to choose between two variants and its accuracy is therefore higher.

Identifier and score: For the identifier and score classes, only a single variant is generated as this proved to be the best solution during the development. The results are thus the same in all columns.

5.4 Error analysis

This section looks at particular classes and errors we observed on the test datasets. Specifically, we will look at why the correct variants were not reachable, even in the oracle variant.

Cardinal, ordinal and decimal numbers: For cardinal, ordinal and decimal numbers, most of the variants not generated correctly are due to generating a different inflection variant. This means that the generated normalization is valid, but the human annotators preferred another valid variant (see Table 5.3).

Text	Annotation	Generated	Translation
118	st**o**.NOM osmnácti.GEN	st**a**.GEN osmnácti.GEN	one hundred and eighteen
302.	tř**i**.NOUN st**a**.NOUN druhá.ADJ	tř**í**.ADJ st**á**.ADJ druhá.ADJ	three hundred and second
11,1	jedenáct cel**á**.SG jedna	jedenáct cel**ých**.PL jedna	eleven whole one

Table 5.3: Examples of differences in annotated and generated inflections for cardinal, ordinal, and decimal numbers. The first example shows the differences in inflection - NOM is nominative and GEN is genitive. The second one is an example where either the whole expression or only a part of it can be inflected - NOUN stands for noun and ADJ for adjective. The third example is an expression where it is grammatically correct to use both the plural (PL) and the singular (SG).

Another frequent cause of observed errors for decimal numbers is the lack of a variant that would read "0.5" as "půl" (half) instead of "celá pět" (point five), see Table 5.4.

Text	Annotation	Generated	Annotation - translation	Generated - translation
118	dva a půl	dva celé pět	two and a half	two whole five

Table 5.4: Example where the decimal part of a number could be pronounced as "půl" (half).

Text	Annotation	Generated	Annotation - translation	Generated - translation
7:00	sedmou	sedm nula nula	seven	seven zero zero
19 Kčs	devatenáct korun československých	devatenáct Československých korun	nineteen Czechoslovak crowns	nineteen Czechoslovak crowns

Table 5.5: Example of time and money classes where other permissible variants are generated.

Time, money and measure: For the time and money classes, we also observed that our system generated permissible variants, just not the ones chosen by annotators; see Table 5.5 for examples.

For time expressions, we discovered a few cases where the correct semiotic class was not recognized due to the use of a rare form (see Table 5.6). For measure and money expressions, the same problem showed up as an non-recognition of less common or non-standardly written units or currencies (see Table 5.6).

Text	Annotation	Generated	Annotation - translation	Generated - translation
12h11'	dvanáct hodin a jedenáct minut	jedna dva h jedna jedna	twelve hours and eleven minutes	one two h one one
1 hp	jedna koňská síla	jeden hp	one horsepower	one hp
5 zl	pět zlotých	pět zeptolitrů	five zlotys	five zeptoliters

Table 5.6: Examples of non-recognized time, measure unit, and currency. In the last case, note that for zloty, the correct symbol is zł.

Roman numerals: There is also a problem with the recognition of Roman numerals. This occurs, for example, in the cases of "Karel I. Veliký" (Charles I. the Great), where our rule system cannot distinguish this situation from "I. P. Pavlov", where "I." is, of course, an abbreviation of this person's first name and not a digit. Another example is the first-class road name "I/22", which should be pronounced "jedna lomeno dvacet dva" (one slash twenty-two) and not as "I lomeno dvacet dva" (i slash twenty-two).

Identifier and mixed: As already mentioned, the same errors occur for the identifier and mixed classes. The typical error is reading the identifier or mixed word as the corresponding cardinal number; see Table 5.7.

Text	Annotation	Generated	Annotation - translation	Generated - translation
9101	devadesát jedna nula jedna	devět tisíc sto jedna	ninety-one zero one	nine thousand one hundred one

Table 5.7: Example where a tram identifier number should be digit-by-digit, but is instead normalized as a cardinal number.

5.5 Comparison with Google TTS

We decided to compare our system with the commonly used Google TTS. The choice fell on Google, because it supports Czech and at the same time is freely accessible for short texts.

For a rigorous comparison, it would be useful to retrieve a larger number of sentences from Google TTS, at least around 250, after they have been normalized, but before the synthesis takes place. Since we do not have this option, we came up

with two proposals, both of which perform speech synthesis using Google TTS in a first step. This is followed by either listening and manually transcribing the audio outputs or using ASR to convert the sentences into text form. Unfortunately, the first option is too time-consuming, in addition to the listening time, we have to take into account that we probably have to hear the sentence more than once, or listen to it more slowly. The second one is error-prone due to the high number of non-standard words.

Therefore, we decided on a qualitative comparison, where we selected twenty sentences in which our system, according to the automatic evaluation, makes mistakes. We sent these to Google TTS for input and manually transcribed them into text form.

On these twenty difficult sentences, we observed that both systems erred in recognizing or reading less standard units. Neither system can handle `ot/min` (rpm) or `645 tis.` where `tis.` should be converted to "tisíc" (thousand). Of course the systems differ in many errors, e.g., `hp` is not converted to "koňská síla" (horsepower) in our system, but Google handles it correctly. On the other hand, Google normalizes certain units into their English form, which after running through the Czech synthesizer become completely unintelligible, e.g., `6,120 ly`. For others, even a valid but nonsensical Czech word is created, e.g., `407.8 ppm` is read as "čtyři sta sedm celá osm pum" (four hundred and seven point eight cougars).

For both systems, errors in determining the semiotic class of the token are also visible. In the phrase "line 556001", there should be a reading of the number digit-by-digit by convention, which neither system performed. Similarly, the sentence "jednotka Köln 137.852" (unit Köln 137.852) are not read correctly – as two cardinal numbers separated by a dot – by either system. Our system classifies the dot as a thousand separator and Google as a decimal separator. For the text `2005 12. 17.` our system recognizes the date and reads the word December correctly, while Google reads the two numbers with the period as ordinals.

Overall, both systems seem to normalize the basic semiotic classes well. In the case of the more complex ones, our system produces more grammatical errors, but without changes in meaning, which are more common in the case of Google.

5.6 Possible future improvements

Based on our system's results (Table 5.1 and Table 5.2) and error analysis (Section 5.4 and Section 5.5), we propose possible improvements, at least some of which we plan to address in the future.

Scoring methods. The difference between the oracle and the individual language models shows how much we can improve if we always choose the right option. We can do this in two ways: The first one is reducing the number of options the model have to choose from to make the model's job easier, but by doing so we also reduce the number of tokens that can be correctly normalized. We would therefore prefer to further explore the second option, which is to improve the scoring so that the correct option is selected more often even while maintaining the number of options that cover more possibilities in general. A possible approach may be to finetune the language models.

Measure units coverage. We need to increase the coverage of measure units since during the evaluation, it became clear that the units already included are insufficient, e.g., ot/min (rpm), hp (horsepower) are not processed.

Annotation speedup. Due to the number of different phenomena that may occur during text normalization, we would like to expand the number of sentences in our datasets, possibly adding more specialized datasets. In this respect, it would be helpful to achieve more efficiency in annotating the data, as the speed reported by annotators of the current datasets – 20–40 sentences per hour – is very low. Pre-generation of annotations by the system and subsequent correction by human annotators could help here. Of course, this process may limit the diversity of the obtained datasets, as a human annotator may keep the suggested option in the dataset even though they would read the text differently when not prompted. Therefore, we still aim to collect at least some portion of each dataset from humans without pre-generated options.

Performance improvement. Speeding up the whole process, especially selecting the contextually correct option, is essential for real-time running. In our work, we addressed this problem by reducing the number of options generated. However, further reduction would certainly be appropriate, for example, by changing the ranking module. Another possibility to reduce the latency is training a tagger capable of determining morphosyntactic categories for numerical tokens. This would allow the system to always generate only variants with the correct morphosyntactic properties and thus significantly reduce the number of variants produced.

Conclusion

This thesis aimed to create a module for the Czech number normalization task with the expected use as part of the TTS pipeline. We had successfully fulfilled the goal set and created the Czech normalization module achieving more than 80% sentence accuracy on both Wikipedia and Czech Radio datasets.

We began by presenting the text normalization task and its challenges, with particular attention to morphologically rich languages (Chapter 1). We continued by introducing the theoretical concepts used in the implementation and discussing related work for other languages (Chapter 2).

An essential Chapter 3 describes the architecture of our implemented system, which consists of four steps - Preprocessor, Tokenizer & Classifier, Verbalizer and Ranking module. In addition to a detailed description of each part, we provide a number of examples, a description of the development dataset and the changes made in the division into semiotic classes compared to the theoretical division presented in Section 1.2.

As at least a moderate-sized dataset was required to evaluate our approach and no Czech dataset was publicly available, we created a dedicated data annotation application for text normalization purposes (Chapter 4). Since lack of data can also be a problem in other languages, we publish our application.

Using the application, we have collected the first public Czech datasets for the task: (1) Wikipedia containing 1,484 sentences and 2,586 to-be-normalized tokens and (2) Czech Radio with 398 sentences and 599 tokens needing normalization. Both datasets are published on Github.[5]

Finally, we evaluated our implementation using these datasets, carefully examined the results, and provided a thorough error analysis (Chapter 5). We are aware that the newly developed system still has some shortcomings, which are described in the section on future improvements (Section 5.6).

[5] https://github.com/RuzickaJakub/czech-number-normalization-datasets

Bibliography

Steven Abney. Part-of-speech tagging and partial parsing. *Corpus-based methods in language and speech processing*, pages 118–136, 1997.

Giusepppe Attardi. WikiExtractor. https://github.com/attardi/wikiextractor, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Tyler Baldwin and Yunyao Li. An In-depth Analysis of the Effect of Text Normalization in Social Media. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 420–429. The Association for Computational Linguistics, 2015. doi: 10.3115/v1/n15-1045. URL https://doi.org/10.3115/v1/n15-1045.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.

Eleanor Clark and Kenji Araki. Text normalization in social media: progress, problems and applications for a pre-processing system of casual English. *Procedia-Social and Behavioral Sciences*, 27:2–11, 2011.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Peter Ebden and Richard Sproat. The Kestrel TTS text normalization system. *Nat. Lang. Eng.*, 21(3):333–353, 2015. doi: 10.1017/S1351324914000175. URL https://doi.org/10.1017/S1351324914000175.

Kyle Gorman. Pynini: A Python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 75–80, 2016.

Kyle Gorman and Richard Sproat. Minimally Supervised Number Normalization. *Trans. Assoc. Comput. Linguistics*, 4:507–519, 2016. doi: 10.1162/tacl_a_00114. URL https://doi.org/10.1162/tacl_a_00114.

Kyle Gorman and Richard Sproat. *Finite-State Text Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2021. doi: 10.2200/S01086ED1V01Y202104HLT050. URL https://doi.org/10.2200/S01086ED1V01Y202104HLT050.

Jan Hajic. *Disambiguation of Rich Inflection - Computational Morphology of Czech*. Charles University, 2004. ISBN 978-80-246-0282-0. URL http://press.uchicago.edu/ucp/books/book/distributed/D/bo6029829.html.

Jan Hajic, Jarmila Panevová, Eva Hajicová, Petr Sgall, Petr Pajas, Jan Štepánek, Jiří Havelka, Marie Mikulová, Zdenek Zabokrtskỳ, Magda Ševcıková-Razımová, et al. Prague dependency treebank 2.0. *CD-ROM, linguistic data consortium, LDC Catalog No.: LDC2006T01, Philadelphia*, 98, 2006.

Hany Hassan and Arul Menezes. Social Text Normalization using Contextual Graph Random Walks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1577–1586. The Association for Computer Linguistics, 2013. URL https://aclanthology.org/P13-1155/.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural Comput.*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Navdeep Kaur and Parminder Singh. Conventional and contemporary approaches used in text to speech synthesis: A review. *Artificial Intelligence Review*, pages 1–44, 2022.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001. ISBN 978-0-262-13360-9.

M. McTear. *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2020. ISBN 9781636390321. URL https://books.google.cz/books?id=kGAMEAAAQBAJ.

Tomás Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048. ISCA, 2010. URL http://www.isca-speech.org/archive/interspeech_2010/i10_1045.html.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent Models of Visual Attention. *CoRR*, abs/1406.6247, 2014. URL http://arxiv.org/abs/1406.6247.

Axel H. Ng, Kyle Gorman, and Richard Sproat. Minimally supervised written-to-spoken text normalization. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2017, Okinawa, Japan, December 16-20,*

2017, pages 665–670. IEEE, 2017. doi: 10.1109/ASRU.2017.8269000. URL https://doi.org/10.1109/ASRU.2017.8269000.

Deana Pennell and Yang Liu. Toward text message normalization: Modeling abbreviation generation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 5364–5367. IEEE, 2011. doi: 10.1109/ICASSP.2011.5947570. URL https://doi.org/10.1109/ICASSP.2011.5947570.

Rafal Poswiata and Michal Perelkiewicz. Numbers Normalisation in the Inflected Languages: a Case Study of Polish. In Tomaz Erjavec, Michal Marcinczuk, Preslav Nakov, Jakub Piskorski, Lidia Pivovarova, Jan Snajder, Josef Steinberger, and Roman Yangarber, editors, *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing, BSNLP at ACL 2019, Florence, Italy, August 2, 2019*, pages 23–28. Association for Computational Linguistics, 2019. doi: 10.18653/v1/W19-3703. URL https://doi.org/10.18653/v1/W19-3703.

Adam Przepiórkowski. *Narodowy korpus jezyka polskiego*. Naukowe PWN, 2012.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-text Transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Lucia Specia, Zhenhao Li, Juan Miguel Pino, Vishrav Chaudhary, Francisco Guzmán, Graham Neubig, Nadir Durrani, Yonatan Belinkov, Philipp Koehn, Hassan Sajjad, Paul Michel, and Xian Li. Findings of the WMT 2020 Shared Task on Machine Translation Robustness. In Loïc Barrault, Ondrej Bojar, Fethi Bougares, Rajen Chatterjee, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Alexander Fraser, Yvette Graham, Paco Guzman, Barry Haddow, Matthias Huck, Antonio Jimeno-Yepes, Philipp Koehn, André Martins, Makoto Morishita, Christof Monz, Masaaki Nagata, Toshiaki Nakazawa, and Matteo Negri, editors, *Proceedings of the Fifth Conference on Machine Translation, WMT at EMNLP 2020, Online, November 19-20, 2020*, pages 76–91. Association for Computational Linguistics, 2020. URL https://aclanthology.org/2020.wmt-1.4/.

Richard Sproat and Navdeep Jaitly. RNN Approaches to Text Normalization: A Challenge. *CoRR*, abs/1611.00068, 2016. URL http://arxiv.org/abs/1611.00068.

Richard Sproat, Alan W. Black, Stanley F. Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. Normalization of non-standard words. *Comput. Speech Lang.*, 15(3):287–333, 2001. doi: 10.1006/csla.2001.0169. URL https://doi.org/10.1006/csla.2001.0169.

Richard W Sproat. *Multilingual text-to-speech synthesis*. KLUWER academic publishers, 1997.

Jana Straková, Milan Straka, and Jan Hajic. Neural Architectures for Nested NER through Linearization. *CoRR*, abs/1908.06926, 2019. URL http://arxiv.org/abs/1908.06926.

Paul Taylor. *Text-to-speech synthesis*. Cambridge university press, 2009.

Shubhi Tyagi, Antonio Bonafonte, Jaime Lorenzo-Trueba, and Javier Latorre. Proteno: Text Normalization with Limited Data for Fast Deployment in Text to Speech Systems. *CoRR*, abs/2104.07777, 2021. URL https://arxiv.org/abs/2104.07777.

Daan van Esch and Richard Sproat. An Expanded Taxonomy of Semiotic Classes for Text Normalization. In Francisco Lacerda, editor, *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pages 4016–4020. ISCA, 2017. URL http://www.isca-speech.org/archive/Interspeech_2017/abstracts/0402.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Krzysztof Wróbel. KRNNT: Polish Recurrent Neural Network Tagger. In Zygmunt Vetulani and Patrick Paroubek, editors, *Proceedings of the 8th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 386–391. Fundacja Uniwersytetu im. Adama Mickiewicza w Poznaniu, 2017.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. *CoRR*, abs/2010.11934, 2020. URL https://arxiv.org/abs/2010.11934.

Yi Yang and Jacob Eisenstein. A Log-linear Model for Unsupervised Text Normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 61–72. ACL, 2013. URL https://aclanthology.org/D13-1007/.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. Neural Models of Text Normalization for Speech Applications. *Comput. Linguistics*, 45(2):293–337, 2019. doi: 10.1162/coli_a_00349. URL https://doi.org/10.1162/coli_a_00349.

Glossary

ideogram A graphic symbol representing a concept directly, independent of any particular language or phrasing. 8

non-standard word Words typically not found in dictionary and cannot be pronounced by an application of ordinary "letter-to-sound rules". 37

semiotic class A group of semantically related non-standard words. Examples are cardinal numbers, dates or measures. 4, 5, 7, 9, 10, 12, 16–21, 23, 25, 27, 29–34, 36, 39, 46

Acronyms

ASR automatic speech recognition. 4, 37

FSA finite-state acceptor. 12, 13

GPT-2 Generative Pre-trained Transformer 2. 14, 15, 17, 24

GRU gated recurrent unit. 11

LM language model. 9, 14, 15, 17, 18, 20, 23–25, 32–35, 37

LSTM long short-term memory. 10

NER named entity recognition. 19

NN neural network. 3, 9, 10, 16

NSW non-standard word. 4–8, 11, 12, 19–21, 27, 44

RNN recurrent neural network. 11, 14

SER sentence error rate. 11, 12

TTS text-to-speech. 3–9, 20, 39

WER word error rate. 11, 12

WFSA weighted finite-state acceptor. 12–14

WFST weighted finite-state transducer. 9, 12–14, 17–21

A. Attachments

A.1 Attached files

In Table A.1 the files attached to the thesis are listed. Each of the attached folders contains a `README.md` file containing a description of the corresponding component, including instructions for running it.

Filename	Description
`czech-number-normalization-datasets`	Directory containing the annotated datasets
`tts-normalization`	Directory with source codes for the normalization module.
`text-normalization-annotation-application`	Directory with source codes for the annotation application.

Table A.1: List of attached files.

A.2 Annotation manual

The following manual is part of the annotation application. It describes how to use the application and examples of each semiotic class and its verbalizations. The manual does not include instructions for installing and running the annotator, this information is provided in the documentation for the source files.

Anotační framework pro normalizaci textu - Manuál

Anotační framework pro vytváření anotačních sad pro účely normalizace textu s ohledem na normalizaci čísel.

Předpokládáme, že máte přístup k běžící instanci anotátoru.

Ovládání anotátoru

Přihlášení

1. Otevřete si v prohlížeči anotátor.
2. V pravém horním rohu klikněte na možnost přihlášení.
3. Vyplňte své přihlašovací jméno (přidělené administrátorem).

Nahrání datasetu

1. Na horní liště vyberte záložku *Datasety*.
2. Zvolte možnost *Nahrát dataset* a vyberte soubor s příponou *.txt* obsahující věty určené pro anotaci.
3. Pokud se *Dataset* neobjeví automaticky, klikněte na tlačítko obnovit.

Výběr datasetu

1. Na horní liště vyberte záložku *Datasety*.
2. Všechny zadané datasety jsou zobrazeny v seznamu společně s celkovým počtem vět a počtem již anotovaných vět.
3. Anotování započnete kliknutím na ikonu start (|--> svislá linka s šipkou směrem vpravo).
4. Po vybrání datasetu se přeneseme do anotačního okna, na větu, která je v pořadí tolikátá, kolik je již anotovaných vět. Předpokládá se postupný průchod od první věty do poslední.

Anotace

1. Máme otevřený vybraný dataset.
2. Pod lištou máme pořadí věty v datasetu a název dataset. Dále níže je editor s originální větou, tlačítka pro pohyb v anotátoru. Případně i již vybrané části věty (tokeny) s normalizovanou formou a kategorií, do které patří.
3. Myší označíme část textu, která vyžaduje normalizaci.
4. Po výběru se nám otevře okno, kde vyplníme, jak vypadá znormalizovaná forma a vybereme kategorii.
5. Uložíme změny, které se hned označí v původní větě a vybraný výraz se zadanými podrobnostmi se přidá do seznamu výrazů vyžadujících normalizaci.
6. Po vybrání všech výrazů změny uložíme, kliknutím na tlačítko uložit.
7. Pokračujeme na další větu.

Export dat

1. Na záložce *Datasety* klikneme na tlačítko *Stáhnout data*.
2. Všechny datasety i uložené anotace ve formátu JSON jsou staženy na počítač ve formě zip archivu.

Import dat

1. Na počítači máme uložen zip archive vyexportovaný z annotatoru.
2. Na záložce *Datasety* klikneme na tlačítko *Nahrát dataset*.
3. Vybereme zip archive z našeho počítače.
4. Datasety i anotace z archivu jsou nahrány na náš účet.

Semiotické třídy / Kategorie

Následující přehled zmiňuje všechny kategorie, které je možné anotovat. V případě, že žádná možnost neodpovídá, pak zvolte možnost *jiné* (*other*), které je v přehledu zmíněno úplně na konci. U každé kategorie jsou uvedeny příklady výrazů z dané kategorie a příklady verbalizací - jak by se daný výraz přečetl.

Základní funkce (Base features)

- základní číslovka (cardinal number)
  - 1; 100; 35 500; 432, +14, -20
  - `35 -> třicet pět`
- řadová číslovka (ordinal number)
  - 1.; 200.; 321.; *Kapitola* 8 (*Kapitola* není součástí čísla)
  - `321. -> tří stý dvacátý první`
- desetinné číslo (decimal number)
  - 1,123; 3,14; -14,1; 2.71 (with dot not correct in czech but used)
  - `2,5 -> dva a půl`
  - `14,2 -> čtrnáct celých dvě desetiny`
  - `1,123 -> jedna celá sto dvacet tři`
  - `-3.43 -> mínus tři celé čtyřicet tři`
- sekvence číslic (digit sequence)
  - 041
  - `041 -> nula čtyři jedna`
- datum (date)
  - `12.` března; 12.10.2021; 10/12/2022; 2/10
  - `12.10.2021 -> dvanáctého října dva tisíce dvacet jedna`
  - `1/2/1990 -> prvního druhý devatenáct set devadesát`
- čas (time)
  - 13:20; 01:12; 9.45; 5'30"; 3h10m
  - `13:20 -> třináct hodin dvacet minut`
  - `9.45 -> devět čtyřicet pět`
  - `5'30'' -> pět třicet`
- míra (measure)
  - 100 m; 40 km/h; 90°; 2,8 g/cm3; 3×10−6 m/s2
  - `100 m -> sto metrů`
  - `2,8 g/cm3 -> dva celá osm gramů na centimetr krychlový`
- peníze (money)
  - $200; 40 CHF; 30 Kč
  - `40 CHF -> čtyřicet Švýcarských franků`
  - `30 Kč -> třicet korun českých`

Rozšířené funkce - čísla (Extensions - connected to numbers))

- římská číslovka (roman number)
  - *Karel* V.; V; LCD (*Karel* není součástí čísla)
  - `V. -> pátý`
- telefonní číslo (telephone number)
  - +420 604 586 567;+49 211 5684962;0211 5684962
  - `+420 604 586 456 -> plus čtyři sta dvacet šest nula čtyři pět osm šest čtyři pět šest`
- skóre (score)
  - 4:4; 10:15;
  - `4:4 -> čtyři čtyři`
  - `10:15 -> deset ku patnácti`
- s příponou (with-suffix)
  - 13roční; 12stupňový; 2x; 2krát; 32bitových; 1,7x
  - `13roční -> třinácti roční`
  - `1,7krát -> jedna celá sedm krát`
- identifikátor (identifier)
  - AK-47; NGC6542; M-IG-48-2; 1080i50; F-1
  - `M-IG-48-2 -> M IG čtyřicet osm dva`
  - `747 -> sedm čtyři sedm`
- smíšené (mixed)
  - x220
  - `x220 -> x dva dva nula`
- rozsah (range)
  - 2019-2022; 40Hz–20kHz; 1913/14; 1917/1918
  - `2019-2022 -> dva tisíce devatenáct až dva tisíce dvacet dva`
  - `40Hz-20kHz -> čtzřicet herzů až dvacet kiloherzů`
- matematický výraz (mathematical expression)
  - (x-1) / (x^2)
  - `(x+1) -> levá závorka x plus jedna pravá závorka`
- chemický výraz (chemical expression)
  - CH4 + 2 O2 → CO2 + 2 H2O; CO2; (CuSO4 · H2O); ribulóza-1,5-bisfosfátu
  - `C3H8 + 5O2 -> 3CO2 + 4H2O -> c tři h osm plus pětkrát o dva se přemění na třikrát c o dva plus čtyřikrát h dvě o`

Rozšířené funkce - jiné (Extensions - not connected to numbers)

- zkratka (abbreviation)
  - atd.; sv.; s.r.o.; př. n. l.
  - `atd. -> a tak dále`
  - `sv. -> svatý`
  - `př. n. l. -> před naším letopočtem`
- akronym (acronym)
  - ČR; NATO; ISO/IEC
  - `NATO -> nato`
- URL
  - https://example.com; www.root.cz
  - `https://example.com -> HTTPS dvojtečka lomeno lomeno example tečka COM`

    - `www.root.cz -> WWW tečka root tečka CZ`
- email (email)
    - jiri+dva@protonmail.ch
    - `nekdo+dva@matfyz.cz -> nekdo plus dva zavináč matfyz tečka CZ`
    - `karelb.nbk@cuni.cz -> karelb tečka n b k zavináč cuni tečka CZ`
- adresářová cesta (pathname)
    - /root/user; C:\windows\user\
    - `C:\windows\user -> disk C adresář windows user`
    - `/root/var/log -> absolutní cesta root var log`
- vynechané/ignorované (ignored)
    - :-
    - `: ->` (normalizováno na prázdný řetězec)
- název symbolu (verbatim - special symbol)
    - †, *
    - `† -> umrtí`
    - `* -> narozena`
- jiné (other)