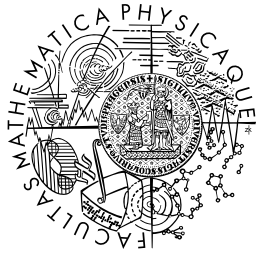

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Tomáš Sourada

Automatic inflection in Czech language

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis:	Mgr. Rudolf Rosa, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

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 ..... date ....

Author's signature

I would like to thank my supervisor Mgr. Rudolf Rosa, Ph.D. for his support, great supervision, time flexibility, and his readiness to help me with every problem I encountered.
I am especially grateful to my consultant Mgr. Jana Straková, Ph.D. for her invaluable advice on training and tuning neural models.
I appreciated the FAQ section on website of Mgr. Milan Straka, Ph.D. on the use of computational clusters and his lectures on deep learning.
I would like to thank Martin Kavka for providing me with database of neologisms, prof. Panevová for providing me with a book about ASIMUT system, and Miroslav Pecka for providing me with script from sklonuj.cz. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.
Most of all, I would like to thank my family for their support, Spanish family Alkain-Nieto with whom I spent most of my time working on this thesis, and Lupita Papáčková for her support, care and simply for the fact that she exists.

Title: Automatic inflection in Czech language

Author: Tomáš Sourada

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Rudolf Rosa, Ph.D., Institute of Formal and Applied Linguistics

Abstract: This thesis focuses on the task of automatic morphological inflection of Czech nouns, specifically in out-of-vocabulary (OOV) conditions (inflecting previously unseen words). We automatically extracted a large dataset suitable for training and evaluation in the OOV conditions. We also manually built a real-world OOV dataset of neologisms. We developed three different systems: a retrograde model performing a variation of kNN algorithm, and two sequence-to-sequence (seq2seq) models based on LSTM and Transformer. Compared to an available rule-based inflection system sklonuj.cz and standard SIGMORPHON shared task baselines, our seq2seq model reaches the best results in the standard OOV conditions. Moreover, it achieves state-of-the-art results for 6 out of 16 development languages from SIGMORPHON 2022 shared task data in the OOV evaluation (feature overlap) on large data condition. On the real-world OOV dataset, the retrograde model outperforms all neural models and is competitive with a non-neural SIGMORPHON baseline. We release the inflection system with seq2seq model as a ready-to-use Python library. It could serve as a complement to the state-of-the-art dictionary-based inflection system MorphoDiTa as a back-off for OOV words, especially once extended to other parts of speech.

Keywords: inflection, declension, morphological inflection, morphology, Czech language, natural language generation, out-of-vocabulary words

Contents

Introduction

Inflection Inflection is a process of word formation where a base word form is modified to express grammatical categories. Dealing inflection in some languages is quite simple: for example in English the nouns have the same form in all cases and only change usually by adding "-s" when changing from singular number to plural. Nevertheless, in morphologically rich languages such as Czech, the inflection is much more profound: the nouns usually have up to 7 distinct forms in singular and 7 in plural.

Morphological inflection in the terms of natural language generation is the task of generating the inflected form subject to the lemma (base form) and the morphological information about the desired form.

Ready-to-use systems The ready-to-use systems for Czech inflection are either dictionary-based or rule-based. Dictionary-based systems provide perfect inflection for the words present in their dictionary but no inflection for the other words. An example is MorphoDiTa[1] [Straková et al., 2014], which contains a very large morphological dictionary. On the other hand, rule-based systems[2] are able to inflect any given word and therefore could be used as a back-off for out-of-vocabulary words (OOV) when using a dictionary-based system. However, they have relatively low accuracy due to insufficient coverage of the hand-written rules they consist of.

Academic work In the academic world, the inflection has been extensively explored in the recent years. Since 2016, SIGMORPHON has held an annual shared task on morphological inflection, including datasets for up to 103 languages [Cotterell et al., 2016; Cotterell et al., 2017; Cotterell et al., 2018; McCarthy et al., 2019; Vylomova et al., 2020; Pimentel et al., 2021; Kodner et al., 2022]. The increasingly prevalent approach is the employment of the sequence-to-sequence (seq2seq) neural network architectures [Sutskever et al., 2014] based on LSTM [Hochreiter and Schmidhuber, 1997] or the Transformer

[1] `https://lindat.mff.cuni.cz/services/morphodita/`

[2] e.g. `https://sklonuj.cz/`

[Vaswani et al., 2017], using the encoded lemma and morphological information about the inflected form as the input for the network, and the inflected form itself as the output. Especially the Transformer-based systems seemed to almost completely master the task, achieving outstanding results, especially when the training data was plentiful [Wu et al., 2020; Pimentel et al., 2021]. However, it has been shown that the performance was artificially inflated by the presence of training lemmata in the test dataset, and that the systems achieve rather poor results when tested on previously unseen inputs (OOV words) [Liu and Hulden, 2021; Goldman et al., 2022]. The last year's iteration of the shared task [Kodner et al., 2022] evaluated the performance of submitted systems in the OOV conditions, but Czech dataset was not included.

Our approach We produce a large dataset of Czech inflected nouns[3] aimed at inflection evaluation on unseen words. The dataset is produced by automated preprocessing, filtering and splitting of the pre-existing MorfFlex dataset [Hajič et al., 2020] into pair-wise lemma-disjoint training, development and test sets. Moreover, we manually create a small dataset of real-world OOV words (neologisms).

We develop three different systems, all data-driven. The first one is the retrograde model, which is dictionary-based and adapts the approach of k-nearest-neighbors algorithm: when given a lemma, search the database for a word that is most similar (has the longest common suffix) and inflect the lemma according to it. The second and the third one follow the standard neural approach using sequence-to-sequence architecture based on either LSTM [Hochreiter and Schmidhuber, 1997] or Transformer [Vaswani et al., 2017].

We adapt the systems to our OOV setting and extensively tune them. Then we evaluate them and compare to one existing ready-to-use system, and to SIGMORPHON shared task baselines [Pimentel et al., 2021] on our datasets. Our systems either outperform the other evaluated systems or perform comparably.

We train and evaluate one of our best setups on SIGMORPHON 2022 shared task data (18 languages, Czech not included) [Kodner et al., 2022] in the large training data condition, and achieve either competitive or better results in the OOV evaluation condition (feature overlap) compared to the submitted systems.

Finally we address the lack of a reliable morpho-guesser for Czech by

[3]For simplicity we focus on nouns only. However, we hope that the systems we develop and the approach we adapt is extendable to other parts of speech.

releasing one of our best systems as a ready-to-use python library[4].

Main contributions The main contributions of this work are as follows:

- building large train-dev-test split of an existing dataset for out-of-vocabulary conditions for Czech nouns
- manually creating small real-world out-of-vocabulary dataset
- adapting the standard-architecture models to our setting and extensively tuning them
- evaluation of established systems on our dataset and comparison with our models
- achieving state-of-the-art results in the OOV evaluation condition in 6 out of 16 development languages from SIGMORPHON 2022 shared task in large data condition
- releasing a ready-to-use morphological guesser for Czech nouns with state-of-the-art performance, that could work as a complementary system for MorphoDiTa [Straková et al., 2014] (especially once extended to other parts of speech).

Contents In Chapter 1 we introduce the necessary concepts from the linguistic background and define the task. In Chapter 2 we describe the basics of neural network architectures, list the ready-to-use inflection systems for Czech and summarize the academic approaches to the task.Chapter 3 describes the datasets and the process of their creation. In Chapter 4 we list the baseline systems we use for comparison and the 2 systems we develop: the retrograde model and the seq2seq model. Chapter 5 defines the evaluation setup and reports all relevant experiments we conducted. In Section 5.5 we compare all the systems on our test datasets and in Section 5.6 we report our results on the SIGMORPHON's 2022 dataset. Chapter 6 describes the structure of the attached repository and provides a detailed description of the most important parts of the code. Finally in Chapter 7 we present the inflection library and describe how to install and use it.

[4] `https://github.com/tomsouri/cz-inflect/releases/tag/BP_official`

Chapter 1

Task and background

1.1 Linguistic Background

To be able to talk about the task of inflection properly, we first describe what inflection is and remind the most important terms. Since this work deals with Czech language, we usually translate the terms to Czech, introducing the corresponding variants in *italics*.

1.1.1 Nouns and their morphological categories

Noun is the part of speech that includes words which refer to people, places, things, ideas, or concepts. Nouns may act as subjects of the verb, objects of the verb, indirect object of the verb, or object of a preposition (or postposition)[1].

Examples of nouns in Czech are *žena* (woman), *muž* (man), *hrad* (castle) or *moře* (sea).

The most important morphological (grammatical) categories of nouns are gender, number and case. In Czech, gender (*rod*) can be feminine (*ženský*), masculine animate (*mužský životný*), masculine inanimate (*mužský neživotný*) or neuter (*střední*). Number (*číslo*) can be singular (*jednotné*) or plural (*množné*).

Case (*pád*) is a grammatical category determined by the syntactic or semantic function of the noun[2]. There are seven different cases in Czech, usually referred to by its number (1 to 7) or by so called case question (*pádová otázka*) that can be used to ask and get answer in the corresponding case.

The list of all cases in Czech, together with corresponding case questions and with examples in singular and plural is in Table 1.1.

[1] `https://glossary.sil.org/term/noun`

[2] `https://glossary.sil.org/term/case`

	Case	*Case question*	Singular	Plural
1	Nominative *Nominativ*	who? what? *kdo? co?*	a man *muž*	men *muži*
2	Genitive *Genitiv*	whom? of what? *koho? čeho?*	(of) a man *muže*	(of) men *mužů*
3	Dative *Dativ*	to whom/what? *komu? čemu?*	(to) a man *muži*	(to) men *mužům*
4	Accusative *Akuzativ*	whom/what? *koho? co?*	(I see) a man *muže*	(I see) men *muže*
5	Vocative *Vokativ*	addressing *oslovení*	man! *muži*	men! *muži*
6	Locative *Lokál*	about whom/what? *(o) kom? (o) čem?*	(about) a man *muži*	(about) men *mužích*
7	Instrumental *Instrumentál*	with whom/what? *(s) kým? (s) čím?*	(with) a man *mužem*	with men *muži*

Table 1.1 Case in Czech with corresponding case number, case question and examples in singular and plural. We can notice that in English, the word remains the same in all cases in the same number, changing only from singular to plural, while in Czech the word form is different in different cases.

Negation

Negation is sometimes considered as a separate morphological category in Czech. This is probably caused by the fact that the negative variant of a word is formed by simply adding the prefix "ne-" (e.g. *pohodlí* (comfort) and *nepohodlí* (discomfort)).

The inflected forms of the negative variant of a lemma are considered as inflected forms of the original lemma, and the original lemma has 14 negative and 14 affirmative forms.

1.1.2 Inflection

Inflection is a process of word formation where a base word form is modified to express grammatical categories. In Czech it is usually expressed by adding or changing suffixes.

A simple example of inflection in English is forming a plural form usually by adding suffix *-s/-es* to the singular form (e.g. *robot/robots*, *process/processes*).

While English nouns remain the same when changing the case, Czech nouns change, leading to the total of 14 forms of one noun (7 cases in singular, 7 in plural), most of which are usually distinct.

Some nouns have 7 forms only: pluralia tantum, having plural only (e.g. *kalhoty* (trousers), *nůžky* (scissors) or geographical proper nouns, such as Pardubice or České Budějovice), and some other geographical proper nouns that do not form a plural (e.g. Rovensko).

Some nouns are *inflexible*, which means that they do not change the form when changing the case and number. Those are usually loan words (e.g. *filé* (fillet) or *žervé* (cream cheese)).

Inflection of nouns (and also adjectives, pronouns, numerals) can be called declension, while inflection of verbs is called conjugation.

Czech term for inflection *(ohýbání)* is not used much, usually we speak about declension *(skloňování)* and conjugation *(časování)*. However, in English it is more common to speak about inflection in general, therefore we will refer to noun declension as to inflection.

1.1.3 Lemma, Tag, Inflected form, Paradigm

In inflection we usually deal with a triple lemma, tag, inflected form.

Lemma (pl. *lemmata*, although the form *lemmas* has been used in the recent years extensively), sometimes called citation form, is the base form of a word, which is then modified by inflection. In terms of morphological categories, lemma is usually equal to its nominative case in singular (or to its nominative case in plural, for pluralia tantum).

Tag represents a bundle of morpho-syntactic features. A tag can be represented in several ways. An example of a tag can be `S3`, which is a tag representing a word form in singular (S) in dative case (3).

Inflected form or *word form* (also *wordform*) is the word (token) created by inflection. Usually we want it to be exactly determined by a lemma and tag.

However, in Czech there are lemmata that have multiple (usually 2) different forms for a given tag, e.g. masculine animate nouns that belong to inflectional paradigm *pán* (man/master), *muž* (man) or *soudce* (judge). For dative (3rd case) and locative (6th case) in singular they have two different forms, both equally correct (*pánovi/pánu*, *muži/mužovi*, *soudci/soudcovi*).

Paradigm table is the full set of inflected forms for a lemma. Each *paradigm cell* is associated with a specific morphological feature bundle. An example of a complete paradigm table is in Table 1.1, where the lemma is *muž* (a man) and the considered morphological features are number and case. For simplicity, we omitted the other possible different forms *mužové*, *mužovi* (described in the previous paragraph) in the table.

When a lemma inflects the same as some other lemma, we say that the other lemma is a *paradigm* for the lemma. Children at the primary school

learn some typical paradigms and how to inflect them and then apply the knowledge to inflect other, previously unseen words, just by identifying to which of the known paradigms the word belongs. An example of paradigm taught at the primary school is *hrad* (castle). If we know the forms of *hrad* (*hrad, hradu, hradu, hrad, hrad, hradu, hradem, ...*) and we know that *hrad* is the inflectional paradigm for *dub* (oak), we can create the forms of *dub* by adapting the suffixes of the forms of *hrad*: *dub, dubu, dubu, dub, dub, dubu, dubem, ...*

1.1.4 Homonyms

Homonyms in general are words that are written the same but have different meaning. When the difference in meaning causes that two lemmata that are equal in their base form have different paradigm tables, we talk about morphological homonyms.

In our work we do not consider semantic-only homonyms (those that have different meaning but their paradigm tables are equal, e.g. *kolej* (track/hall of residence)) as different lemmata. When we mention homonyms in our work, we mean the morphological homonyms. An example of morphological homonyms are some words in masculine gender that can be both animate and inanimate and they are the same in the base form, such as *vyhledávač*. This can be both a search engine (inanimate) or a spotter/seeker (animate; a person that searches something). Another example would be word *rada* which can be either of masculine (councillor) or feminine (advice) gender.

1.1.5 OOV words

Out-of-vocabulary words (*OOV words*, *unseen* or *previously unseen* lemmata) are generally *novel* lemmata, words seen for the first time. When we have a system performing some language task, OOV words are the words not present in the data that were used for training and adjusting such system. Therefore the fact whether a word is OOV or not depends on the used data. We can talk about general OOV words, words that are expected to be unseen. Some common OOV words can be neologisms, proper nouns (e.g. names or surnames), misspelled words or (in some languages) words with removed diacritics.

	HRAD	
hrad ⟶	hrad	hrady
	hradu	hradů
	hradu	hradům
	hrad	hrady
	hrade	hrady
	hradu	hradech
	hradem	hrady

	LINGEBRA	
lingebra ⟶	lingebra	lingebry
	lingebry	lingeber
	lingebře	lingebrám
	lingebru	lingebry
	lingebro	lingebry
	lingebře	lingebrách
	lingebrou	lingebrami

Table 1.2 The task: for a given lemma, generate all inflected forms (whole paradigm table). An example for a standard paradigm *hrad* (castle) and an OOV word *lingebra* (short for *lineární algebra*, linear algebra). Left column of the each paradigm table corresponds to singular, right column to plural.

1.2 Task

This work aims at the task of morphological inflection, sometimes called morphological generation, inflection generation or morphological inflection generation. It is a task from the field of natural language generation.

We specifically focus on inflection of OOV nouns in Czech.

Our inflection task therefore is: given a previously unseen Czech noun lemma, generate 14 forms corresponding to 7 different cases and 2 different numbers (see Table 1.2 for an example).

This is a special variant of the paradigm table completion problem (paradigm cell filling problem), which is a task of completing an incomplete inflection paradigm table. In our conditions, incompleteness means that only the base form (the lemma itself) is present. There are 14 desired forms, because Czech nouns have usually 7 forms both in singular and plural.

A broader task approached by this work is producing a ready-to-use system capable of inflecting previously unseen lemmata of Czech nouns. To achieve the best possible performance, we do not restrict the size of data used for building (training and adjusting) the system.

We want the system to predict correctly not only the individual forms, but also the whole paradigm tables, and we evaluate it accordingly.

Consequently, the data provided for training should consist of whole paradigm tables, as well as the data for evaluating the system (see Figure 1.1).

The system should be as data-driven as possible. We would like to omit hand written rules completely. In consequence, the system should be theoretically extendable to other languages.

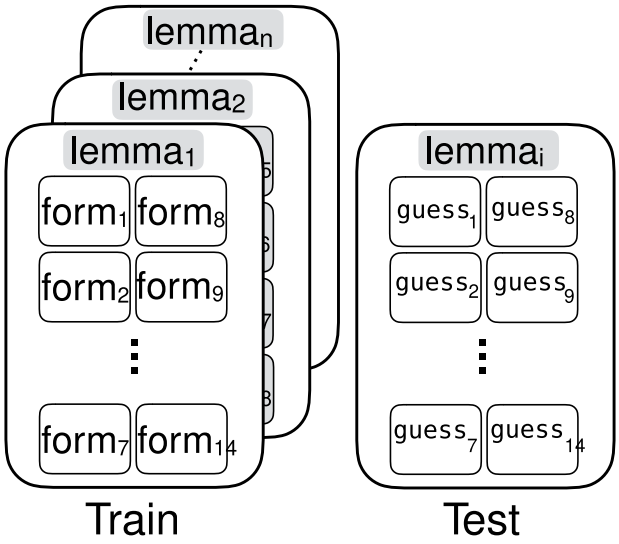

Figure 1.1 The task during training and evaluation. Left: in the training condition, we have a lot of complete paradigm tables. Right: in the test condition, we have only the lemma itself and the system has to produce all inflected forms. (figure adapted from Cotterell et al. [2017])

Since the lemma is given without any context and additional linguistic information, our system will not be capable of correctly dealing homonyms.

For simplicity we only consider case and number as the relevant morphological categories. We do not consider negation as a morphological category and we treat lemma and its negated variant as two different lemmata.

Also for simplicity we always want to generate 14 forms, one form per one paradigm cell, even for lemmata that have several different forms for some cells that are equally correct, and also for lemmata that have some stylistic variants for some paradigm cells.

We do not want the system to decide whether a form exists (e.g. non-existent singular forms for pluralia tantum). We rather want it to predict 14 forms for every given lemma.

Chapter 2

Related work

In this chapter we report important topics from related work: neural network architectures, ready-to-use inflection systems, and academic approaches to inflection.

2.1 Neural network architectures

In this section we describe the basics of neural network architectures that are broadly used for morphological inflection. The following subsections about neural networks, recurrent neural networks, seq2seq models and transformers were automatically generated by chatGPT[1] and we post-edited them[2].

2.1.1 Neural networks in general

Neural networks are an important class of machine learning models. They consist of interconnected layers of artificial neurons (units), organized into input, hidden, and output layers.

The training process of neural networks involves two main stages: forward propagation and back-propagation. During forward propagation, the input data is fed through the network, and each neuron performs a weighted sum of its inputs, followed by the application of an activation function. This process continues layer by layer until the output layer produces a prediction. The predicted output is then compared to the ground truth, and the difference (loss) is calculated.

[1] ChatGPT May 24 Version

[2] The original conversation is included in the attachment, and also available at this link: `https://chat.openai.com/share/76fcc235-8830-4f3d-9328-5888874abf52`

The goal of training is to minimize the loss by adjusting the network weights, while processing the training data. This is achieved using an optimization algorithm, such as stochastic gradient descent (SGD) [Ruder, 2016] or Adam [Kingma and Ba, 2017], which iteratively modifies the parameters to minimize the loss based on the gradient of the loss function with respect to the network's parameters.

Several factors play a crucial role in training neural networks. The *batch size* determines the number of training samples processed together before updating the parameters. The number of *training steps* refers to the total number processed batches. An *epoch* represents a complete pass through the entire training dataset. The number of epochs can be calculated as the number of train steps multiplied by the ratio of batch size and the size of the training dataset (see equation (2.1)).

$$\#\text{ epochs} = \frac{(\text{batch size}) \cdot (\text{train steps})}{\text{train size}} \tag{2.1}$$

Model selection is a technique of choosing the best-performing model from a set of trained models (checkpoints) that are evaluated on a development dataset.

2.1.2 Recurrent neural networks (RNNs)

Recurrent neural networks (RNNs) are a type of neural networks that excel in processing sequential data, such as natural language. Unlike traditional feedforward neural networks, RNNs have connections between neurons that form directed cycles, allowing them to retain information about previous inputs.

The most important RNN architectures are Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Unit (GRU) [Chung et al., 2014], both able to learn long-term dependencies. GRU is faster than LSTM due to simpler architecture. However, LSTM outperforms it on large datasets with short sequences [Yang et al., 2020].

2.1.3 RNN-based seq2seq models with attention

Sequence-to-Sequence (*seq2seq*) models [Sutskever et al., 2014] in general have the ability to process variable-length input sequences and generate corresponding output sequences of different lengths.

The architecture of RNN-based seq2seq model with attention (see Fig-

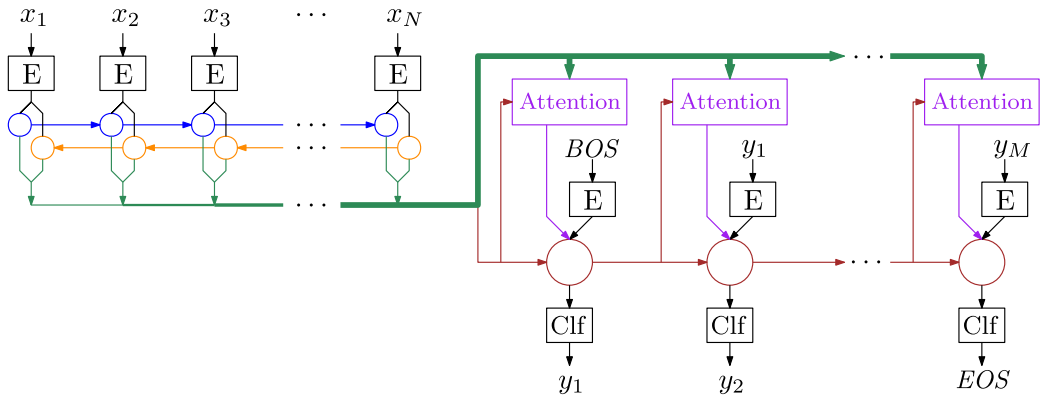

Figure 2.1 Schema of RNN-based seq2seq architecture with attention: encoder with single bidirectional layer and decoder with a single layer. Clf is the softmax classifier used to obtain the output tokens from the network hidden state.

ure 2.1[3]) consists of an *encoder* and a *decoder*, both built using RNNs. The encoder (left part of the figure) processes the input sequence (x_1 up to x_N in the figure), represented as a sequence of character embeddings (computed by the embedding matrix, "E" box in the figure), to generate a sequence of *context vectors*, one for every element of the input sequence (green arrows in the figure).

The decoder (right part of the figure) uses *attention mechanism* [Luong et al., 2015] (violet "attention" box in the figure) to focus on different parts of the input sequence (context vectors) dynamically, attending to the most relevant information for each output token, and generates the output sequence step by step (y_1, y_2 up to end-of-sequence (EOS) token in the figure). At each time step, the decoder's hidden state is updated based on the previous hidden state (left-to-right red arrows), the attended context vectors (violet arrows), and the previously generated output token (embedded by the embedding matrix - the "E" box).

Embeddings are dense vector representations of the input tokens (words or characters) that capture semantic and syntactic properties. They are typically learned jointly with the model during the training process. In case that the source and target vocabularies do not differ much, it is possible to use shared embeddings between the encoder and the decoder (in the schema in Figure 2.1 it would mean that the "E" box in the encoder part is the same as in the decoder part).

[3] `https://ufal.mff.cuni.cz/~straka/courses/npfl114/2122/slides/?09`, slide 11

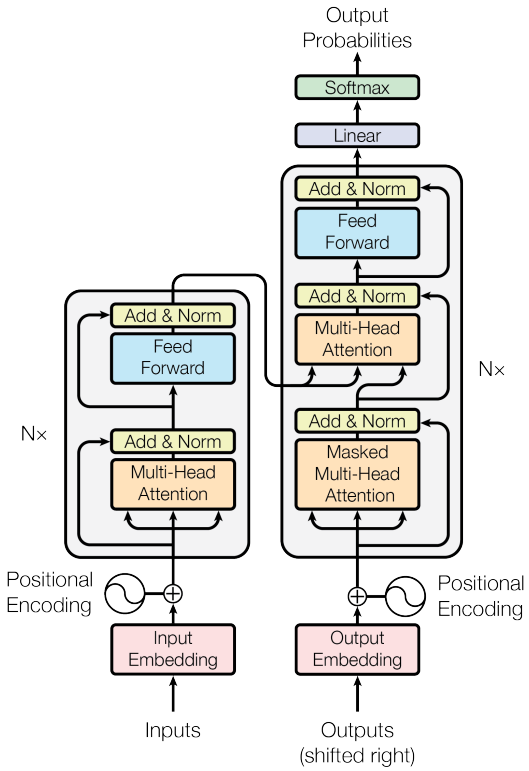

Figure 2.2 Transformer architecture [Vaswani et al., 2017]

2.1.4 Transformer

The Transformer [Vaswani et al., 2017] is a special type of seq2seq model. Unlike RNN-based seq2seq models, the Transformer relies primarily on self-attention mechanisms, eliminating the need for recurrent connections and allowing for parallel processing of input sequences.

The core components of the Transformer architecture are the self-attention mechanism and the feed-forward neural network layers, organized into an encoder and a decoder.

The self-attention mechanism allows the model to weigh the importance of different positions in the input sequence when generating each output token. It enables the model to capture dependencies between all positions in the sequence simultaneously, effectively modeling long-range dependencies.

A schema of the architecture is in Figure 2.2. The model has several important hyperparameters:

- The number of layers (N in the figure) - determines the depth of the model and its capacity to capture complex relationships.

- The hidden layer size - the dimensionality of the model's hidden representations.
- Attention heads and their count - Attention heads allow the model to focus on different aspects or subspaces of the input sequence. Increasing their count improves the model's ability to capture diverse dependencies. (see Multi-Head Attention in the figure)
- The size of the feed-forward layer - determines the dimensionality of the intermediate representation in the Transformer's feed-forward neural networks. (see Feed Forward in the figure)
- The embedding vector size - determines the dimensionality of the input token embeddings. (see Input Embedding in the figure)
- Positional encoding - encodings added to the input embeddings, providing the model with information about the order of the tokens, allowing the Transformer to capture the sequential nature of the input sequence. (see Positional Encoding in the figure)
- Dropout - is a regularization technique. It randomly sets a fraction (based on the dropout parameter) of the activations to zero during training, which encourages the model to learn more robust representations.
- Attention dropout - applying dropout to the attention scores during self-attention calculations. It helps prevent attention from focusing too strongly on specific positions, promoting more diverse and balanced attention patterns.

For further information about how the Transformer works please refer to Vaswani et al. [2017].

2.2 Inflection systems

There are generally two types of inflection systems: (i) ready-to-use systems, possibly publicly available on web, accessible to general public, and (ii) experimental systems, pushing the limits of morphological inflection.

In the following sections we review both types. We report the linguistic module of ASIMUT information retrieval system [Králíková and Panevová, 1990] as a special case, because it had been a ready-to-use system, yet is no longer available for usage.

2.2.1 Ready-to-use systems

There are several ready-to-use inflection systems for Czech available on web.

We divide them according to the main approach to inflection they take, into rule-based and dictionary-based systems.

Rule-based systems

- Sklonuj.cz[4] - we use it as a baseline for comparison
- Ucitel.net[5]
- Aztekium.pl[6]

Based on the information provided by the web maintainers (we asked them by email) we can conclude that all these systems work mostly the same: they explicitly list the ending segments of lemmata together with ending segments of corresponding forms, trying not to rely on any database. To be able to correctly inflect common words that would be inflected incorrectly, they include a list of exceptions with their complete paradigm tables.

If a lemma is not present in the exception database, the guesser part of the system matches the lemma with a specific paradigm based on the ending segment, and generates forms according to the paradigm. Thanks to the guesser they can inflect any given word. But on the other hand, the lists of exceptions are usually too limited, and the classification to paradigms too rough, resulting in relatively low accuracy of the systems. Moreover, they are all focused on inflection of nouns, adjectives, pronouns and numerals, but completely omit inflection of verbs, which we believe is because the paradigm tables of Czech verbs are completely different and much larger.

Dictionary-based systems

- NLP services of Masaryk University[7] [Šmerk and Rychlý, 2009] - use a morphological dictionary (approx. 3.4 million lemma-tag-form entries)
- MorphoDiTa[8] [Straková et al., 2014] is a morphological analyzer, morphological inflection generator, tagger and tokenizer with state-of-the-art results for Czech. Its generation part is based on a large morphological

[4] `https://sklonuj.cz/`, Czech only
[5] `https://www.ucitel.net/online-nastroje/sklonovani`, Czech only
[6] `http://aztekium.pl/sklonovani`, Czech only
[7] `https://nlp.fi.muni.cz/languageservices/#gen`
[8] `https://lindat.mff.cuni.cz/services/morphodita/`

dictionary MorfFlex [Hajič et al., 2020] (approx. 125 million lemma-tag-form entries), and therefore covers significantly more wordforms than the previous system. However, it also does not contain any guesser for OOV words.

Compared to rule-based systems, dictionary-based systems implement only the list of exceptions, completely missing the guesser part. They focus on having extensive coverage of words, and on effective storing of the dictionary together with fast search in it.

In contrast to the listed rule-based systems, they also provide inflection of verbs.

Nevertheless, they miss the guesser part - they perform perfect inflection of words present in the dictionary, but for unseen lemmata they do not do anything.

2.2.2 ASIMUT - inflection in Czech

One of the very early systems addressing automatic inflection specifically in Czech was ASIMUT system [Králíková and Panevová, 1990]. It focused on information retrieval in Czech documents. The authors introduced a linguistic module for inflection (of nouns, adjectives, pronouns, numerals and verbs in Czech), which allowed to search for occurrences of all forms of a given lemma, without the need of explicit listing of the forms.

The main concept is similar to that of rule-based ready-to-use systems: based on the ending segment of lemma decide how the word inflects. They defined list of abstract paradigms (tables of changing suffixes) and developed a sophisticated algorithm that would determine the specific paradigm for a given lemma. Moreover, it would generate all its possible stems and assign one of them to every paradigm cell. Finally, it would combine the stems with the suffixes from the assigned abstract paradigm table to produce the inflected forms. Unlike the listed rule-based systems, they were also able to inflect verbs.

They utilized a retrograde dictionary [Slavíčková, 1975] to define the decision rules of the algorithm and to conclude whether a rule is specific enough or needs to be refined to be able to unambiguously determine the paradigm for all words. For example, they checked that "-ý" as the ending segment of a lemma "*is so unambigous that all the 14,000 words having this ending and included in the retrograde dictionary of Slavíčková [1975], with the single exception of* úterý *(Tuesday), belong to a single declensional paradigm (it is the ending of the "hard" adjectives such as* mladý *(young), of substantivized adjectives such as* vrátný *(janitor), and of adjectival pronouns*

and numerals, such as který *(which),* nějaký *(some) or* druhý *(second))*" [Králíková and Panevová, 1990, pp. 18].

In addition, they developed a variant for text without diacritics, yet facing much more challenges due to highly increased amount of ambiguities of ending segments.

Although this inflectional module improved the performance of the information retrieval system a lot by eliminating the necessity of listing all the forms of a lemma by the user, it has several drawbacks: it is not always possible to unambiguously determine the paradigm; exceptions must be added because of the small scope of the retrograde dictionary; too coarse classification (too little paradigms); does not work as reliably for verbs (too much ambiguity of end segments of base forms of verbs).

2.2.3 Experimental systems

In this section we summarize the most relevant academic approaches to the task of inflection, in particular the systems competing in SIGMORPHON shared tasks.

Early approaches

The early approaches to automatic inflection generation were very diverse. Some of them focused on extracting transformation rules and learning a statistical or machine learning model to decide which rules apply.

Dušek and Jurčíček [2013] proposed a method for statistical morphological generation aimed at robustness to unseen inputs. They utilize lemma-form edit scripts based on the Levenshtein string distance metric [Levenshtein, 1965]. They extract the edit scripts from the training data and train a multiclass logistic regression classifier to predict the edit script for a given lemma and features. They treat separately changes at the beginning of the word and changes in the stem or at the end of the word. They evaluate the system on six languages included in the CoNLL 2009 Shared Task data sets [Hajič et al., 2009], including Czech. These are just tokenized texts, and therefore contain a lot of inflexible words (adverbs, participle, conjunctions, and also punctuation). They report the performance not only on the whole evaluation set for each language, but also on a subset containing forms unseen in the train set (OOV words), to be able to measure the robustness of the system to unseen inputs. However, they do not report the performance on unseen lemmata. They report a significant improvement compared to a dictionary baseline (which simply remembers the train set and outputs the lemma itself for forms unseen in the train data).Therefore they conclude that their system

is beneficial - at least as a back-off for unseen forms - even if a large-coverage morphological dictionary is available.

Durrett and DeNero [2013] solve a slightly different task, whole paradigm table prediction. They automatically acquire orthographic transformation rules of paradigms from Wiktionary data[9] (complete paradigm tables) and train a discriminative sequence model to apply the transformation rules correctly to unseen base forms. They evaluate the system on held-out Wiktionary data (complete paradigm tables, in this case unseen lemmata), reporting results for three languages and two parts of speech, obtaining satisfying results, concluding that they built a model that can be used in any language where a substantial number of example inflection tables is available.

The Rise of RNNs

Faruqui et al. [2016] were one of the first to try to adapt the RNN encoder-decoder architecture to the task of inflection, inspired by the great success of that architecture in machine translation. They feed the encoder with individual characters of the input lemma to encode it into a vector, and then decode the vector one character at a time to obtain the inflected form. They tried two different settings: (1) train an individual encoder-decoder for each target tag, (2) train a single encoder for all input lemmata, and then train an individual decoder for each target tag. They report evaluation on several languages, obtaining better or comparable results to previous state-of-the-art systems.

In 2016, the first shared task on morphological inflection was organized [Cotterell et al., 2016] and started a sequence of SIGMORPHON inflection Shared tasks. They included 3 subtasks, one focused on general task of inflection (given lemma and tag, produce the form) and the other two on reinflection (given a form, produce another form). Czech was not included in the 10 languages covered by this iteration of shared task. The best performing non-neural system [Sorokin, 2016] extracted the complete set of abstract paradigms and reduces the problem to multi-way classification. Nevertheless, RNN-based neural systems were the clear winner of the task, especially in high data conditions. It was shown that pre-extracting edit operations is not likely to achieve top-level performance.

Aharoni et al. [2016] introduced a method to bias the seq2seq model to copy individual characters (motivated by the fact that different forms in a paradigm table usually share most of the characters), making it easier to generalize, which was especially important when working with small datasets.

[9] `http://en.wiktionary.org`

They achieved promising results, taking the 2.-3. place in the shared task.

The all-around best performing system in the shared task was Kann and Schütze [2016]. In contrast to Faruqui et al. [2016], they train a single encoder-decoder for all target tags. More specifically, they use the bidirectional-RNN encoder-decoder with attention [Bahdanau et al., 2016]. To allow the usage of a single encoder-decoder for all tags, they concatenate the target tag symbols and the source lemma characters to obtain the input for the encoder. The target characters are produced one-by-one by the decoder. They also make use of special tokens to mark the start and the end of the sequence (both in source and in target). They perform only a few experiments for hyperparameter tuning (hidden layer size, embedding size and the way of initialization) and to compare different input representations. An improved system [Kann and Schütze, 2016] adds an automatic correction method for the outputs based on edit trees.

The next iteration of the shared task [Cotterell et al., 2017] included two subtasks: the general inflection task, and paradigm table completion. Almost all participants used GRU or LSTM with attention, with the common strategy of feeding in the input lemma character by character, along with the morphological subtags of the desired form. The systems were evaluated on 52 languages including Czech, and in three different data conditions (low, medium, high). The results confirmed that the encoder-decoder architecture performs strongly when the training data is plentiful. Kann and Schütze [2017] and Bergmanis et al. [2017], both based on the 2016's winning system [Kann and Schütze, 2016] and adding some data augmentation methods, were amongst the best performing systems for both the tasks and all languages. Kann and Schütze [2017] achieved especially good results in the task of paradigm completion. An alternative approach [Zhou and Neubig, 2017] made use of additional unannotated data utilizing the multi-space variational encoder-decoder.

The 2018's iteration of the shared task [Cotterell et al., 2018] introduced data from 103 languages, including Czech. Some of the competing systems focused on the lower data condition, in which the encoder-decoder models are known to perform worse due to data sparsity. The winner in that conditions made use of sequences of edit-operations, and learned to predict them using RNNs [Makarov and Clematide, 2018]. Some other systems reused the successful RNN encoder-decoder approach from previous years, improving its performance by creating artificial training data to bias the model towards copying.

The 2019's iteration of the shared task [McCarthy et al., 2019] was quite different, focusing on cross-lingual transfer to perform inflection in low-resource languages, and to the task of complete morphological tagging

of a sentence. Some of the approaches utilized in the task were again RNN encoder-decoder (with attention) and learning edit actions.

The Rise of Transformers

Although the Transformer [Vaswani et al., 2017] had become a popular architecture for sequence-to-sequence problems shortly after being introduced, outperforming RNN-based seq2seq models in various word-level NLP tasks (most significantly, in neural machine translation), the first work successfully applying it on the task of morphological inflection was Wu et al. [2020], reporting state-of-the-art results on the SIGMORPHON shared task data from 2017 [Cotterell et al., 2017].

They adapted the architecture used for machine translation by reducing its capacity. They also observed that batch size plays a crucial role when using the Transformer on character-level tasks, showing the performance improves steadily as the batch size increases. They finally fixed it at value 400, which is especially large, considering that the used training data contains only 10k examples.

The next iteration of the shared task [Vylomova et al., 2020] reacted to the recent success of the Transformer architecture, employing it as one of the baselines. The task provided data for 90 languages (Czech not included) and aimed at generalization across languages. A lot of systems used the Transformer, utilizing data hallucination (generating synthetic training examples that mimic the patterns and characteristics of the original data, thereby increasing the size and diversity of the training set) and other augmentation techniques, and for low-resource languages also multi-lingual training. Amongst the four winning systems, two of them were multilingual, RNN-based with attention, and two were monolingual, based on Transformer (e.g. Liu and Hulden [2020], developing an ensemble of three Transformers, all with the same architecture, but with different input data format).

The Fall of Transformers on Unseen Lemmata

However, Liu and Hulden [2021] showed that the Transformer almost completely fails in the task of inflection when asked to inflect a previously unseen lemma.

They criticize the common-practise test conditions used in 2016-2020 SIGMORPHON shared tasks [Cotterell et al., 2016; Cotterell et al., 2017; Cotterell et al., 2018; McCarthy et al., 2019; Vylomova et al., 2020], where there is an uncontrolled overlap between lemmata in the training and test set. They use 2018 shared task development and test data for six languages

to evaluate the approaches again. They build new corresponding training datasets by extracting full paradigm tables from UniMorph, such that the lemmata are not present in the dev nor test dataset. The built training datasets have the same size as the 2018 datasets (10k forms), but have zero lemma-overlap. In addition, they utilize the 2020 shared task data for some low resource languages, rebuilding the provided dataset (obtained by joining train, dev and test) in the same ratio, assuring empty lemma-overlap.

They evaluate the previously successful system [Liu and Hulden, 2020] using the original training data and the new training data, reporting a huge gap in performance between these two conditions.

In addition, they suggest several techniques of hallucination methods improving the performance even in the zero-lemma-overlap condition.

Nevertheless, we think that the training datasets they utilized to compare the performance were not directly comparable, because beside having different lemma overlap, the new training datasets covered far fewer distinct lemmata in total (because they were created from the whole paradigm tables).

The 2021 SIGMORPHON shared task [Pimentel et al., 2021] focused on generalization across typologically diverse languages. The organizers probably did not have enough time to react to the presented findings about the bad properties of the common-practise testing with lemma-overlap [Liu and Hulden, 2021]. Accordingly, the train-dev-test split they performed was again on the instance level, with uncontrolled lemma overlap.

Nevertheless, they at least evaluated all the systems (4 Transformer baselines and 2 submitted systems, both based on RNN encoder-decoder with attention, each with different data augmentation method) also on the subsets of test set containing unseen lemmata only, reporting the drop in systems' performance on previously unseen lemmata. They report that the drop is larger for Transformer-based systems. On the other hand, systems that utilized some data augmentation method did not lose that much in the performance in the OOV condition.

However, this analysis was only possible in the languages where the test set contained at least some unseen lemmata. In consequence, for Czech and several other languages, the evaluation on unseen lemmata is not included.

Goldman et al. [2022] investigate more the problem of lemma overlap between training data and test data. They propose a split-by-lemma method for the train-test split, performing such re-split on the data from SIGMORPHON's 2020 shared task. They re-evaluate 3 of the 4 top-ranked systems (2 Transformers, one RNN-based encoder-decoder) from that iteration of shared task, comparing the performance on the original data and on the

new data, without lemma overlap. They reveal that the new splits lead to a decrease of 30 points (averaged over the 3 systems, for all 90 languages). They conclude that generalizing inflection to unseen lemmata is far from being solved, although it is a desired function in practise. Nevertheless, they show that on high-resource languages with 40k training examples or more, the systems do not lose much with the new split.

Finally, SIGMORPHON's 2022 shared task [Kodner et al., 2022] emphasizes generalization along different dimensions, evaluating test examples with unseen lemmata and unseen features separately. The data consist of 33 languages (30 development languages and 3 surprise languages). They provided two different training data conditions: small and large, yet for some languages the large training dataset was missing due to insufficient amount of data. Czech language was not included; the most similar to Czech was Polish and Slovak.

They split the test dataset into 4 parts: with lemma overlap (the lemma was seen in the training data, but the feature was not), feature overlap (the feature was seen in the training data, but the lemma is novel - that is our task), both overlap (both the lemma and the feature were seen in the training data) and neither overlap (both the lemma and the feature are novel), and evaluate the systems in all the conditions separately.

Some participated systems used Transformers, but not all (some also made use of Finite State Transducers [Merzhevich et al., 2022], alignments or RNNs for edit sequences). The overall best-performing system was based on Transformer [Yang et al., 2022]. It made use of hallucination methods, lemma copying and other methods to generalize to unseen lemmata.

Generally, the performance on novel features was much worse than on novel lemmata. Some systems even seem to possess a significant ability to generalize to unseen lemmata. The results also show that the models perform substantially better with more training data.

In the time of writing this thesis, SIGMORPHON's 2023 shared task has not been evaluated yet and the paper is not available. Nonetheless, from the task description[10] it seems that it does not focus on the generalization on novel lemmata nor novel features, but rather on the interpretability of morphological reinflection models.

[10] `https://github.com/sigmorphon/2023InflectionST`

2.3 Conclusion of related work

The state-of-the-art system for morphological generation in Czech, MorphoDiTa [Straková et al., 2014] does not have a guesser for OOV words, and there is no publicly available system that would complement it and work reliably.

Although SIGMORPHON included Czech in several iterations of its' shared task, the performance of systems on unseen lemmata has never been evaluated systematically.

The best performing models in the shared tasks usually employed seq2seq models based on RNN or Transformer, frequently utilizing some data augmentation techniques, which brought the greatest benefit when the training data was scarce, yet did not help much when the data was plentiful.

Chapter 3

Data

To build a data-driven inflection system, we obviously need some data: a training data to build or train the model, development data to decide, which parameters are better, and also to compare different models, and test data, to finally evaluate the best models.

3.1 Why to build our own dataset

The aim of our work is to perform inflection of OOV words (especially nouns) in Czech. Therefore we need a Czech morphological dataset, and we need the development and test set to be lemma-disjoint with train set (ideally those three sets should be lemma-disjoint).

And since we want to build best system possible, there is no need to limit the size of the data, as long as we have enough computational resources.

It would be useful to select a dataset already used in some research, ideally in some iteration of the SIGMORPHON's shared tasks, to be able to compare our system performance with some other relevant approaches. Nevertheless, as far as we know, in the previous work there was no dataset complying our conditions. The only Czech dataset obtained by lemma-split (ensuring that there are no common lemmata in train and test set) used in the SIGMORPHON Shared tasks was the one for Task 2 in 2017 [Cotterell et al., 2017], which was used for paradigm table completion (the train set contains 200 full paradigm tables, and the test set 50 incomplete paradigm tables). Although the report of 2021 Shared task [Pimentel et al., 2021] examines the performance of the systems on previously unseen lemmata, curiously, in the Czech test dataset there were no unseen lemmata. And finally, Czech was not included in the 2022's iteration of the shared task [Kodner et al., 2022], which investigated the performance on OOV lemmata.

Although we could make use of the dataset from 2017, or perform a disjoint-lemma-resplit of a dataset from another iteration of the shared task, we consider their size too limited. The shared tasks mostly focused on performance in low-resource conditions. However, Czech is no longer low-resource language with regard to morphological inflection. And as we already emphasized, our main aim is to build a good system. Therefore we decide not to constrain our system to only use a small dataset to learn, and we build our own datasets.

Motivated by the fact that in our task, OOV words are those that are not present in the training morphological dictionary (we build a complementary system to MorphoDiTa which lacks a morphological guesser for generation), and by the tremendous coverage of inflected forms it has, we decide to utilize MorfFlex [Hajič et al., 2020] to build 3 large lemma-disjoint datasets: training, development and test set. Besides, it is relatively simple to process MorfFlex thanks to its format.

To be able to evaluate the final models in the real-world OOV conditions, we decided to build a dataset some true OOV words.

We considered several options of what to use as the real OOV words: misspelled words, words with removed diacritics, proper nouns, and neologisms.

We think that misspelled words should not be treated as OOV words. It would be better to automatically correct them and then inflect as standard dictionary words.

As for words with removed diacritics (i.e., from undiacritised variant of a lemma create undiacritisied variants of the inflected forms), it would be an interesting work to investigate. Králíková and Panevová [1990] built a separate inflection system for words without diacritics and faced several difficulties caused by high number of ambiguous lemmata in the undiacriticised variant. Generally we would like to decide ahead whether the generated forms should be with or without diacritics. Therefore it would be reasonable to build two distinct models: one for general inflection (the one we are building in this work) and one for inflection of lemmata without diacritics, which would be built probably in the same way and from the same data, only with removed diacritics. However, it would be important to consider that this would lead to more ambiguities in the data as pointed out by Králíková and Panevová [1990]. Consequently, even undiacritised words are not appropriate representatives of real-world OOV words (for the undiacritised model they would not be OOV).

Proper nouns could be an interesting class to inflect, yet we consider them too specific in the meaning (names of people, places etc.), probably also too specific in the morphological properties. Moreover, lot of proper nouns are included in dictionaries (MorfFlex contains more than 150k proper-noun

lemmata).

In consequence, we finally choose to build the real-world OOV dataset of neologisms, which are words that cannot be included in a dictionary by their very nature.

3.2 Design decisions

The task of noun inflection in our setting is defined quite precisely: given a lemma, provide 14 inflected forms. Since we would like to evaluate not only the correctness of single forms, but also of the whole paradigm tables, it makes sense to build the development set and the test sets from complete paradigms. However, sometimes it can be convenient to evaluate the models also on the training data (e.g. to detect whether a neural network is overfitting or not). This leads us to accepting a specific requirement: all data should be composed of whole paradigm tables. In other words, if we have one line per inflected form, we have 14 lines for each lemma.

In the following subsections we discuss the special cases that may be problematic regarding this requirement.

3.2.1 Non-existent forms in the data

Nonetheless, from the very beginning we were aware that there are some lemmata that simply do not have 14 forms (e.g. pluralia tantum forming plural only). Omitting the entries corresponding to the missing forms would lead to problems when evaluating the accuracy on whole paradigm tables, and it would not follow the requirement stated in the previous paragraph. Hence we decided to introduce a special token representing the non-existent forms. Further in the text we denote the "special token representing the non-existent form" simply as *non-existent form*. The fact that the presence of non-existent forms in the training data could cause problems is discussed in Section 4.3.2.

3.2.2 Multiple forms in a paradigm cell

Moreover, in Czech there are sometimes natural ambiguities with regard to a specific form in a paradigm cell, as discussed in Section 1.1.3, e.g. *páni* vs. *pánové* for nominative in plural of lemma *pán* (man, gentleman). These ambiguities mean that for some specific paradigm cells, there are several possible forms and we cannot definitely decide which one is more appropriate, because it is dependent on the style and context. In general, it would be reasonable to include all possible forms in the training data, with the objective

of covering the heterogeneity of the language, so that the models would be able to generate all forms. Furthermore, it is even more logical to include the ambiguities in the testing data, in order to avoid penalization of some models generating correct forms, yet different from those present in the data.

Nonetheless, to simplify the process of dataset building and dealing with the data during splits and training, we decide not to include multiple forms in one paradigm cell, while being aware of all the drawbacks mentioned above and knowing that in future work, the dataset could be built more carefully.

The only exception we make is for the real-world OOV dataset. The main reason is that for OOV words, it is even more difficult to decide which of the possible forms is more appropriate. For example for dirty words (which are common in our data), a form that would ordinarily be labeled as non-standard or informal, seems more appropriate than the standard form, because dirty words are usually used in informal speech. Besides, we consider the true OOV dataset hard enough for prediction by itself and we do not want to increase the difficulty by penalizing the models for generating correct, yet different forms.

To make this exception feasible, we introduce a separation token, which can only be used in evaluation data (in the gold targets) and which separates multiple different forms in one paradigm cell.

It is interesting to note how this problem was dealt with in some previous research. In the 2021 and 2022 iterations of SIGMORPHON's shared tasks [Pimentel et al., 2021; Kodner et al., 2022] they report completely removing all instances with duplicated lemma-tag pair. This includes not only forms of lemmata with multiple possible forms for a specific paradigm cell, but also entries corresponding to homonymous lemmata. Both these cases appear the same in the data (duplicated lemma-tag pair), but semantically are different. However, since in Czech some particular word classes allow consistently more than one form in a specific single paradigm cell (refer to Section 1.1.3), we think that by dropping all these examples they are omitting a specific structure from the data. For evaluating of inflection systems it is not necessarily an issue (and as such, it is suitable for the needs of a shared task). Yet for building an inflection system capable of providing inflections for any word, completely omitting the examples (i.e., both variants if there are more than one) is probably not the best approach.

3.3 MorfFlex - the morphological dictionary

MorfFlex CZ 2.0 [Hajič et al., 2020] is a Czech morphological dictionary originally developed as a spelling checker and lemmatization dictionary. It

babička	NNFS2-----A----	babičky
babuška_,h_^(babička)	NNFP4-----A----	babušky
Písek-2_;G	NNIP6-----A----	Píscích

Table 3.1 MorfFlex entry examples: *babička* (grandma), *babuška* (granny) and *Písek* (a town located in the south of Czechia).

contains more than 125 million entries. Each entry consists of a lemma-tag-wordform triple. For each wordform, full inflectional information is coded in the tag (not only the traditional morphological categories, but also some semantic, stylistic and derivational information).

The entries cover lemmata that occur in real Czech texts (Prague Dependency Treebank – Consolidated 1.0 (PDT-C 1.0) [Hajič et al., 2020]), e.g. Czech words, foreign words and loan words, abbreviations, proper nouns, isolated letters, parts of words numbers and other non-alphanumeric characters. Apart from standard Czech, the paradigms contain also non-standard variants (even defective forms, misspelling, typos), which are marked in the tag [Mikulová et al., 2020].

First, we briefly describe the data format of MorfFlex. Then we explain how we process and filter the data, obtaining one huge dataset complying with the conditions described in Section 3.2. Finally we report how the train-dev-test split is performed, and discuss the limitations of our approach to process the data.

3.3.1 MorfFlex data format

The MorfFlex dictionary is a flat list of lemma-tag-wordform triples (see some examples in Table 3.1). It follows the principle of unique analysis (lemma and tag together uniquely identify the wordform). To achieve this, lemma numbering is used to capture homonyms that are morphologically different, and tag numbering is used to capture different types of wordform variants [Mikulová et al., 2020].

All information in the following sections about lemma, tag and wordform are taken from the complete technical specification and explanation of the format [Mikulová et al., 2020]. It is really complex and we mention the most important parts only.

Lemma

Lemma represents the whole paradigm. An example lemma is `Písek-2_;G`. Lemma has two parts. The first part, *lemma proper* (`Písek-2` in the example),

is "*a unique identifier of the paradigm. Usually it is the base form of the word* (we call it raw lemma in context of MorfFlex), *possibly followed by a number distinguishing different lemmas with the same spelling but different formal morphological behavior*" [Mikulová et al., 2020]. Those are morphological homonyms. Amongst noun raw lemmata, almost 2% are homonymous.

The second part, *additional information*, is optional and can contain name or style label, and variant or derivation information, etc. In the example, it is `_;G`, indicating that it is a capitalized word of type geographical name.

We can notice that in the example of lemma *babička* in Table 3.1 the lemma only consists of the base form, omitting the number and additional information.

We call the whole lemma (lemma proper together with the additional information) *rich lemma*.

Tag

Tag (positional tag) is a string consisting of 15 characters. An example of a tag is `NNFP7-----A---6`. Each position in the string corresponds to one morphological category. The most important categories for us are:

- part of speech (1st position, `N`=noun)
- number (4th position, `P`=plural, `S`=singular, `X`=any)
- case (5th position, `1-7`, `1`=nominative, `7`=instrumental, `X`=any)

Another noteworthy categories are:

- gender (3rd position, `F`=feminine, `N`=neuter, `I`=masculine inanimate, `M`=masculine animate, and some combinations)
- negation (11th position, `A`=affirmative, `N`=negated)
- variant (15th position, `-`=basic variant, `1-4`=standard variant, `5-9`=non-standard variant).

Please note that the stated values are just the most frequent ones, not the only possible values for that categories.

We can see that in MorfFlex, negation is treated as a separate morphological category. With a few exceptions, all negation forms in MorfFlex are formed by adding prefix "ne-".

Wordform

The third part of each entry is the word form itself. It is always the word form without any additional information.

Data	# lemmata	# forms
whole MorfFlex	1,055,757	125,348,901
nouns only	459,554	10,824,178
basic-variants only	459,554	9,268,047
no negations	459,554	6,268,561
no unusual tagsets	449,044	6,144,804
completed paradigm tables	449,044	6,286,616

Table 3.2 Data filtering: the numbers of lemmata (homonyms counted multiple times) and inflected forms during the data filtering.

3.3.2 Data filtering and conversion

We start with the whole MorfFlex, an extremely large set of individual entries, each identified by the unique (rich lemma, tag) pair, and our goal is to obtain data as described in Section 3.2. That is, list of raw noun lemmata, each associated with a list of 14 forms, either present or non-existent.

It is important to keep in mind that there are homonymous lemmata – for those we will have multiple entries (paradigm tables) in the final dataset.

To achieve our goal, we perform several processing and filtering steps, while following the principles formulated in Section 3.2 (see data counts after each relevant step in Table 3.2):

1. remove all but noun entries
2. convert rich lemma to the corresponding raw lemma (e.g. `babuška_,h_^(babička)` to `babuška`), while preserving the association of each wordform with the specific lemma
3. remove all non-basic-variant forms
4. remove all negation forms
5. remove lemmata with unusual sets of tags
6. complete all incomplete paradigm tables.

The reason for only keeping nouns is straightforward since our work focuses on nouns. We elaborate more about the other steps in the following paragraphs, mentioning the limitations caused by each step. It is important to note that the decisions about the process of data filtering were made without exploring the effect of each step to the performance of a system trained and evaluated on the corresponding data. We believe that the effect is low or even negligible. Nevertheless, in future work it would be better to perform deeper analysis.

Keep homonyms (Rich lemma to raw lemma)

The step of converting rich lemma to raw lemma is connected with the decision of keeping or dropping entries for homonyms. We decide to keep them.

Before the conversion, each lemma is different, thanks to MorfFlex format. However, after the conversion, for homonymous lemmata there are multiple paradigm tables with the same base form. This is generally an issue, because it leads to having duplicate lemma-tag pairs with different target form. Having such duplicates in training data can in general be reasonable, and is similar to the problem of ambiguous forms in a paradigm cell (discussed in Section 3.2.2). However, if such examples appear in the evaluation data, it causes that any system that for a specific input (lemma and tag) consistently returns a specific output (target form) cannot achieve 100% accuracy. That is probably the reason why Pimentel et al. [2021] and Kodner et al. [2022] decided to drop all lemma-tag duplicates.

As we deal with the paradigm table as atomic, we can either drop the whole paradigm (ideally for all homonymous lemmata) to get rid of ambiguity, or keep paradigms for all homonyms. We decide to keep them all, with the objective of not removing any significant information from the dataset. To resolve the issue with evaluation, we compare the systems with an oracle system that represents the upper bound of the performance of any system that that for given lemma-tag pair consistently returns the same inflected form.

Non-basic forms removal

As discussed in Section 3.2.2, we decided to include only a single form to one paradigm cell. Therefore we have to resolve the issue of other standard and non-standard forms, present in MorfFlex data.

One possible approach would be creating a special paradigm table when there are multiple forms possible. However, this would lead to excessively copying the forms that have only one possibility, and to artificially inflating the size of the dataset.

To simplify the processing, we remove all non-basic forms (i.e., other forms marked as standard or non-standard).

Negation removal

MorfFlex treats negation as a morphological category (see Section 3.3.1), i.e., the negated forms have the same lemma as the affirmative forms. However, we do not deal with it this way. Our task is the inflection by number and case only - we consider a lemma and its negated form as two different lemmata,

that in general could be both present in the training data as independent lemmata. (Nevertheless, for the purpose of OOV words, if the affirmative lemma was seen in the training data, we would never consider its negative variant as unseen, and vice versa.)

Hence, dealing with negation forms in MorfFlex data could be done in two different ways: (i) adding two paradigm tables to the data for those lemmata that have negation forms - one paradigm for affirmative forms and one for negated forms, with the base form negated too), and (ii) removing the negation forms from data completely

As already discussed (Section 1.1.1), negation forms are created very simply in Czech, by adding prefix "ne-". Therefore we would expect a system that is capable of inflecting a specific lemma to be capable of inflecting its negation too. That plays in favor of the second approach. Moreover, since the affirmative and negative forms are so similar, adding the negative paradigm tables to the data would lead to artificially increasing importance of those words that form negation, compared to the words that do not form it. Besides, it would result in unnecessarily large dataset.

We evaluate the ability of a system trained without negation forms to inflect negated lemmata in Section 5.4.5.

Deficient paradigm tables removal

Removing lemmata with unusual sets of tags was a simplifying step. As a usual set of tags we consider the full 14-tag set, 7-tag sets of singular or plural forms, and then tag sets that we believe correspond to inflexible words (forming nominative in singular only).

We manually inspected the data, examining some of the lemmata with unsual tag sets, seeing that they frequently miss some wordform or they include some extra form. Considering that they are only a small part of the whole dataset, we decided to drop them. (We are aware that by this we reduced the coverage of our dataset. Consequently, it is possible that we consistently miss some particular class of words and our system is neither able to inflect them correctly, nor we can measure the incorrectness.)

Partial paradigm completion

Completing the incomplete paradigm tables is a problem connected with the preceding step. We had to decide how to complete the paradigms that do not have all 14 forms. Following the decisions we made when choosing which paradigms to drop, we fill in the incomplete paradigms as follows: insert non-existent form token to the cells corresponding to singular forms

of pluralia tantum (and also to the cells corresponding to plural forms of lemmata forming only singular). For the inflexible lemmata, we fill in the copy of the lemma to every empty cell.

Again, we are aware that especially the latter leads to some bias of our dataset: specifically bias to copy the lemma. If the corresponding lemmata were indeed inflexible, it is not a problem because their forms are equal to the lemma itself. Yet we were able to manually inspect only a small part of the dataset, hence it is possible that we were mistaken. The other option would be dropping these lemmata, nonetheless we considered that they form too large part of the dataset. Other option would be completing the empty cells with non-existent forms, which would lead to having too much non-existent forms in the data. Nevertheless, it is an option to be considered for future work.

3.3.3 Train-dev-test split

Once we obtain the dataset processed by the steps described in the previous section, we perform a train-dev-test split on the dataset. Based on its size we decide to build a train-set consisting of 360k lemmata and both dev-set and test-set of 44k lemmata.

To truly comply with the OOV-condition of our task, we follow the split-by-lemma method as proposed by Goldman et al. [2022], who criticized the standard random split by form used in the SIGMORPHON Shared tasks (2016-2021), because after using the standard split, there were non-empty intersection of lemmata contained in the train-set and in the test-set; additionally, the size of the intersection was uncontrolled.

We execute the split-by-lemma as a random split, yet satisfying the condition that train-lemmata, dev-lemmata and test-lemmata are all pair-wise distinct. To achieve that, we carefully treat homonyms (lemmata that are equal in the baseform but have distinct paradigm tables) present in the data.

We considered also other options of performing the split by lemma, not only the random one, such as frequency-based, as applied by Cotterell et al. [2017], who split the data based on the frequencies of corresponding tokens in Wikipedia text. However, we focus on OOV words. Without any further research we cannot claim that the best way to model them is to take the most frequent dictionary words, or contrarily the most rare ones. Therefore we chose the random split.

3.3.4 Limitations of data processing

Some of the limitations of our approach to data processing were already discussed in Sections 3.2 and 3.3.2. Here we only provide a comprehensive list and add some more:

1. negations removal
2. non-existent forms in data
3. removal of non-basic forms
4. keeping homonyms
5. omitting lemmata with unusual tag set
6. completing incomplete paradigm tables with copy of lemma
7. We simply use mostly all data present in morfflex to train the model, without any reasoning which data to use and which not. This follows a simplifying assumption that OOV words are similar to all words from dictionary. This does not necessarily need to be true, yet without any further research we cannot reason whether they are more similar to newer words, less frequent, more frequent, etc. After performing some research in this area, it would perfectly make sense to train the model (which would aim at inflection of OOV words only) on a specific subset of the data.
8. We did not remove duplicates (i.e. equal paradigm tables) from the development data before performing all development experiments. It is important to note that in general we could suppose there are no duplicates, because if two lemmata are equal in the base form and have the same paradigm tables (together with their content), they would be only once in MorfFlex. However, we have caused the presence of duplicates, probably by removing the non-basic forms and reducing the scope to case and number. We mention it here, although the counts of duplicates were very low: 145 paradigm tables in train data (from the total of 360k; 0.04%) and 14 paradigm tables in development data (from total 44k; 0.03%). We removed them from the training dataset for experiments in final evaluation. Neither the test set, nor the test-oov dataset (used in final evaluation) contained duplicate paradigms.

3.4 True OOV

To be able to evaluate our model in real-world conditions, we decided to build our own dataset of true out-of-vocabulary words: neologisms. We collect words from a dictionary of neologism Čeština 2.0 (Czech language 2.0).

3.4.1 Čeština 2.0 - about the project

Čeština 2.0[1] is a dictionary that includes brand new words, but also slang, regional or otherwise interesting expressions from the Czech language. Some of them are actually word phrases or existing words that were given a new meaning (e.g. "powerpoint karaoke" with the meaning of reading the whole slides during a presentation). In June 2023, the dictionary contained almost 27k words (and phrases). Users can "like" or "dislike" the published words.

Looking into the most popular ones[2], we can say that a lot of them were derived from dirty words[3] [4], or come from political context[5] (usually with negative sentiment).

According to the web of the project "*it shows that Czech has a flair, a wit and a future. The dictionary is being created by users who are generally any Czech speakers*"[6].

Each entry in the dictionary contains the word or word phrase together with the explanation and usually also an example of usage in sentence or conversation.

We asked the webpage maintainers and they kindly provided us with a significant part of the dictionary (we received a dataset containing all words beginning with 'e' or 'j' on 12th of March, 2022). We are aware that this subset of data cannot be generally viewed as a random subset (and therefore as a representative subset). Nevertheless, we hope that the first character of a lemma does not have a significant influence on the way the word inflects. (This hope is supported by the fact that Czech is mostly suffixing language and that changing case or number usually affects the ending segments only.) Therefore we treat the data subset as a subset representing the whole dictionary without bias.

[1] `https://cestina20.cz/`, in Czech only
[2] `https://cestina20.cz/zebricek/nejoblibenejsi/`
[3] `https://cestina20.cz/slovnik/kurvitko/`
[4] `https://cestina20.cz/slovnik/pracurak/`
[5] `https://cestina20.cz/slovnik/milosekunda/`
[6] `https://cestina20.cz/o-cestine-2-0/`, Czech only

3.4.2 Data filtering and manual inflection

The provided data are in `CSV` format. We start by extracting the words together with their explanation and the example of usage, omitting other information. We remove all word phrases and words that are actually present in MorfFlex. We shuffle the entries randomly and filter them manually, removing all not-noun entries. Then we randomly choose more than 100 entries.

We continue by manually inflecting the words. To make the process simpler and more resistent to typos, we actually generate the whole paradigm tables by a publicly available inflection system[7] and then carefully correct the forms based on our linguistic knowledge, always checking the explanation of the word and the example (to be able to properly classify the word to some morphological categories usually affecting the actual inflected form, such as gender). In case of doubts, we review the Internet Language Reference Book[8]. It is managed by Czech Language Institute of Czech Academy of Sciences, and contains inflected forms for vocabulary words. There we find a standard word such that it would be inflected in the same way (again, according to our linguistic knowledge), and inflect the new word based on the found inflected forms.

If there are more possible forms for a given paradigm cell, we include all of them. If a form corresponding to a paradigm cell does not exist (such as singular forms for pluralia tantum), we use a special token for non-existent form.

It is important to emphasize that the correction and completion of inflected forms were performed manually and thus there is no guarantee that it is completely correct. Moreover, in case of neologisms, the way how it is inflected can be sometimes highly subjective and it is difficult to decide which is the proper form.

3.5 Datasets

After performing all the data processing, we have four datasets (see Table 3.3). Train-set is aimed at training the models, dev-set at (hyper)parameter tuning and deciding which models are used in final evaluation. On the contrary, test-set and test-oov-set are strictly prohibited to be used for anything else but for the final evaluation and models comparison.

[7] `https://sklonuj.cz/`

[8] `https://prirucka.ujc.cas.cz/en`

Set	# lemma	# form	# non-existent forms	Source
train	360k	5.04M	42,973	MorfFlex
dev	44k	616k	5,383	MorfFlex
test	44k	616k	5,257	MorfFlex
test-oov	101	1.4k	35	Čeština 2.0

Table 3.3 Datasets. The four datasets, with lemma (paradigm table) counts, form counts and counts of non-existent forms.

Each pair of the four datasets has zero lemma overlap (a lemma included in one dataset is not included in any other dataset).

We can say that train-set, dev-set and test-set originate from the same distribution (they were sampled randomly from the same source), while test-oov-set comes from a completely different distribution. Test-oov-set aims to represent the real-world condition of OOV words. Therefore we expect the performance of the systems to be lower on this set.

The first three sets were built automatically, while the fourth one was built manually, leading to a significant difference in the size of datasets (test-set contains 440-times more data than test-oov).

All four sets contain incomplete paradigm tables (paradigms where some forms are non-existent). On the other hand, only test-oov-set contains for some paradigm cells multiple different forms; the other three sets contain at most one form per a paradigm cell.

Chapter 4

Approach

In this chapter we describe the main approach we follow. We describe the baseline systems, our retrograde system and the seq2seq models. We discuss the data representaion for the neural systems and briefly describe their architecture.

The approach in general (without the specifics of any model) is as follows. Take the training data and utilize it to extract rules or learn parameters. Then generalize and inflect lemmata unseen in the training data.

We focus on inflection of unseen lemmata, because the inflection of seen (dictionary) lemmata is solved perfectly by MorphoDiTa [Straková et al., 2014] and we try to create a complementary system.

The main difference between our task (and therefore our approach) and the SIGMORPHON shared tasks (and corresponding approaches) is that we focus specifically on nouns in Czech, with the aim of developing the best system possible (no need to restrict the size of training dataset), and our goal is to inflect especially the unseen lemmata.

4.1 Baselines

In this section we present the baseline systems we use for comparison with our models. We involve a simple copy baseline, rule-based ready-to-use system Skloňuj.cz, and three standard baselines from the SIGMORPHON shared tasks.

We cannot use a dictionary-based system (such as MorphoDiTa [Straková et al., 2014]) as a baseline for comparison, because we are building the complement part of such system, the guesser part, focusing on inflection of unseen words.

4.1.1 Copy baseline

Copy baseline completely ignores the training data and treats every given lemma as inflexible. For each paradigm cell, it produces the lemma itself as the inflected form. Its performance represents the lower limit for any reasonable system.

The reason why we expect this baseline system to work at least to some extent is that in Czech, usually at least one inflected form is the same as the lemma itself (most frequently the nominative (1st case) in singular). Words in neutral gender that inflect according to the paradigm *stavení* (building, house) even have 10 forms equal to the lemma (6 in singular, 4 in plural).

This system is not suitable for any real application, we only use it for comparison.

4.1.2 Rule-based Sklonuj.cz

The first reasonable baseline is a rule-based inflection system Sklonuj.cz, publicly available on web[1]. It represents the class of ready-to-use systems that are able to deal with OOV words. We use its offline version, which was provided to us by the authors. For a given lemma it produces 14 forms, 7 in singular and 7 plural. For further information refer to Section 2.2.1.

4.1.3 SIGMORPHON Shared task baselines

To compare our system with some systems from academic works, we utilize the baselines used in SIGMORPHON Shared tasks, namely the non-neural baseline used in the 2022 iteration [Kodner et al., 2022] and two neural baselines used in the 2021 iteration [Pimentel et al., 2021]. The source code of all the baselines is freely available[2].

To be able to run the baselines on our datasets, we need to convert our datasets to the format of SIGMORPHON datasets. They use lemma-tag-form triples, where the tag is a semicolon-separated set of morphological features, where the number of the features in general differs for each entry. Since our dataset deals with number and case only, we decided to encode the tags in the format `number;case`, where `number` is either `S` (singular) or `P` (plural), and case is a number (`1-7`).

It is important to note that all the baseline systems where developed to work in low data condition and on all languages. Our dataset is exceptionally large, compared to the SIGMORPHON datasets (5M train entries, compared

[1] `https://sklonuj.cz/`

[2] `https://github.com/sigmorphon/2022InflectionST/tree/main/baselines`

to 10k entries in high-data condition). We discuss separately for each system whether running it on our data is sensible.

Non-neural

The non-neural baseline from 2022 iteration of SIGMORPHON Shared task is identical to the non-neural baselines used in 2017, 2018 and 2020.

According to the summary of the principle of the baseline [Kodner et al., 2022] it *"first heuristically extracts lemma-to-form transformations; it assumes that these transformations are suffix- or prefix-based. A simple majority classifier is used to apply the most frequent suitable transformation to an input lemma, given the morphological tag, yielding the output form."* For a more thorough explanation see Cotterell et al. [2017].

As it is based on frequencies of transformation rules in training data, we think that running it on a larger dataset is not a disadvantage, at least not with regard to the accuracy, although with regard to the prediction speed it could be.

Neural - transformers

The organizers of 2021 iteration of the Shared task [Pimentel et al., 2021] provided two neural baselines, a standard vanilla transformer [Vaswani et al., 2017] and an input-invariant transformer, an adaptation of the transformer to character-level transduction tasks [Wu et al., 2021] (an identical system was used as a baseline also in 2022). Both models follow hyperparameters of Wu et al. [2021]. The organizers of the task also introduced an optional data augmentation method, which we omit because our data is already plentiful.

The important training parameters are set up as follows: 20k training steps, batch size 400, saving periodically up to 50 checkpoints during the training time (extending the number of training steps to always finish an epoch and only allowing to save a checkpoint at the end of an epoch), and then choosing the checkpoint that performs the best on development data.

This works well on SIGMORPHON datasets: e.g. with 10k training examples, one epoch takes 25 training steps, and we run 800 epochs, choosing then from the 50 checkpoints, which were saved once per 16 epochs.

However, with 5M training examples, we need 12.5k training steps for one epoch, and therefore it runs at most for 2 epochs and finally we can choose from only two checkpoints. By this, the usage of development data is eliminated almost completely. To address this issue a little bit, we perform an elementary hyperparameter tuning, trying to adjust the batch size and number of training steps such that it would fit our dataset better and allow

the use of model selection. To allow more comparison, we report results of both the default system and the one with longer training (larger batch size and higher number of training steps).

4.2 Retrograde model

First of our inflection models is retrograde model, which focuses on simple implementation. It adapts the basic idea of the linguistic module in ASIMUT [Králíková and Panevová, 1990], described in Section 2.2.2 - deciding how the lemma inflects based on its ending segment.

Unlike in ASIMUT, we do not extract the abstract paradigm tables manually. When building the model, we rather save all available words as possible paradigms, and during prediction we automatically extract the paradigm table, which is equivalent to automatically pre-extracting the abstract paradigm tables. Our requires much less human labor and linguistic expertise, and allows extending the model to other suffixing languages without any need of substantial change in the source code. Other important difference is that we do not explicitly deal with changes inside the stem. This is done implicitly by changing the ending segment if it is long enough. Furthermore, since we focus on nouns only, we do not face difficulties with complex inflection tables for verbs.

In simple words, our system could be described as follows: If we have a database of words we know how to inflect, and we receive an unseen lemma, we find the database-word most similar to it and inflect the lemma according to it.

We describe further details of our approach in the following sections.

4.2.1 Assign paradigm and inflect

When building the model, we start with a morphological dictionary that contains complete paradigm tables for all covered lemmata. We save all the lemmata together with their inflection tables in a way such that we can effectively search them based on the suffixes.

When we want to inflect lemma X, we search in the database for lemma A, such that X and A are most similar (have the longest common ending segment), and inflect lemma X according to the paradigm of lemma A.

If there are multiple lemmata A in the dictionary that have the same longest common ending segment with lemma X, we inflect X according to all of them and combine the predictions performing majority vote for each paradigm cell.

lemma X to inflect:	úřad	(1)
database lemma A with longest common suffix:	hrad	(2)
lemma X stem:	úř	(3)
database lemma A stem:	hr	(4)

HRAD	
hr-ad	hr-ady
hr-adu	
hr-adu	...
...	
hr-adem	hr-ady

$\longrightarrow$ (5)

ÚŘAD	
úř-ad	úř-ady
úř-adu	
úř-adu	...
...	
úř-adem	úř-ady

$\longrightarrow$ (6)

ÚŘAD	
úřad	úřady
úřadu	
úřadu	...
...	
úřadem	úřady

Table 4.1 Retrograde model: example of inflection according to a paradigm. Lemma X for inflection: *úřad* (office), found database lemma A with the longest common suffix: *hrad* (castle).

The inflection of lemma X according to paradigm A is performed as follows (see Table 4.1, numbers reference to the table): remove longest common suffix from lemma X (1) and lemma A (2) to obtain X-stem (3) and A-stem (4). Then for each paradigm cell take the corresponding A-form and replace the A-stem by X-stem (5, 6).

This was only a brief summary explaining the basic idea of retrograde model. A thorough description can be found in Chapter 6, Section 6.5.

4.2.2 The model in context

It is interesting that the retrograde model is generally k-nearest neighbors algorithm in discrete space, where the distance is measured by the length of the longest common suffix (the longer it is, the closer two words are). Unlike the standard algorithm, we do not fix k and we rather find all nearest neighbors, that have the same distance from the word. The prediction part is more complex here: instead of simply combining of the labels (values) from the neighbors, we adapt the edits on the lemma suggested by the neighbors.

Also, the approach of our retrograde model is similar to the non-neural SIGMORPHON baseline (see Section 4.1.3). The main difference is that in the retrograde model we expect no prefixing changing when performing inflection, while they consider both prefix and suffix changes. On the other hand, we tune our model to provide fast inference even with relatively large training set. The authors of the baseline claim that it was designed for speed of application [Cotterell et al., 2017], yet the speed is acceptable only when little training data is provided.

4.2.3 Limitations

The model relies on two properties of Czech language: (i) when two lemmata have the same ending segment, they also inflect the same, and (ii) during inflection (by number and case), only the ending segment changes while the rest of the word remains the same. This mostly holds in Czech but does not hold at all in some other languages (e.g. semitic languages).The retrograde model is therefore strongly language dependent and we do not expect it to work in some languages.

4.3 Seq2seq model(s)

Inspired by a significant success of neural architectures in morphological inflection tasks[Cotterell et al., 2016; Cotterell et al., 2017; Cotterell et al., 2018; McCarthy et al., 2019; Vylomova et al., 2020; Pimentel et al., 2021; Kodner et al., 2022], we decided to investigate some approaches using neural networks.

More specifically, we employ the RNN-based encoder-decoder as used by Kann and Schütze [2016], and the Transformer as used by Wu et al. [2021].Both are seq2seq architectures (character-level encoder-decoder, where the length of the input and the target sequence in general differ between entries), both with attention. The main idea is feeding the lemma together with the target tag (each character by character) into the encoder. The encoder produces a sequence of vectors as a representation of the whole input, and the decoder then decodes it and generates the characters of the inflected form one by one. The attention mechanism helps the decoder to "pay attention" to specific parts of the input when generating specific parts of the output. In fact, neither the encoder, nor the decoder work directly with the individual characters (and tags). There is an embedding layer which provides a vector representation of the characters (of the lemma and the tag) to the encoder. Both the encoder and decoder then work with the character embeddings. The embeddings are trained together with the encoder and decoder parts. For further description of the architecture, refer to Section 2.1.2.

To be able to run the experiments smoothly, without the need of implementing the architectures on our own, we make use of a practical toolkit OpenNMT [Klein et al., 2017], which provides easy access to both these architectures. Its main aim is the task of neural machine translation (NMT), however the architectures it implements are not necessarily dependent on the task.

We start by explaining the analogy of the inflection task with the task of NMT, continue by discussing the data representation for the seq2seq models. Then we briefly describe the two architectures we use and their most important hyperparameters.

4.3.1 Adapting of the inflection task to NMT world

We can view each individual character of the lemma as a token in the source language, and each target tag symbol as another token in the source language. By combining the lemma and the tag we obtain a "sentence" in the source language (where the individual characters represent the "words" of the language). Each character of the desired inflected form likewise corresponds to a "word" in the target language.

The task of morphological inflection is then simply translation of sentences from the source language to the target language.

However, there are some important differences between the tasks. For example, in the task of inflection the vocabulary size (the count of different tokens in the "languages") is much lower than in NMT. Therefore, the architectures need to be adapted to the inflection task by reducing their capacity.

4.3.2 Data representation for MT models

To be able to run the inflection task with the MT models, we have to decide how to represent the data.

In the rest of the work, we consider the task of inflection as generating 14 forms for a given lemma. Nevertheless, the approach of using encoder-decoder architecture to produce the whole paradigm table when given the base form has never been used in the previous work, at least up to our knowledge. A common approach is to encode the morphological features of the desired form together with the lemma as the input for the encoder [Kann and Schütze, 2016]. To generate the whole paradigm table, it is then sufficient to encode the lemma multiple times, each time with one desired tag, and generate the forms separately.

We think that generating the whole paradigm table when given the lemma would theoretically be possible with encoder-decoder architecture, yet the network would need to remember a lot of additional information, such as how many forms it should generate and which form is it generating right now in each step of generation. As such, the network would have to spend a lot of its capacity to deal with this technical stuff.

src sequence	tgt sequence
J A B L K O # P 7	J A B L K Y

Table 4.2 Baseline data representation. # is the separator token. P 7 is the morphological tag corresponding to plural (P) and intrumental case (7).

Therefore we build a system that takes a lemma and a tag as the input and produces one form as the output.

Similarly to Kann and Schütze [2016], we use the individual characters of the lemma together with the morphological tag as input. To encode the morphological features we need (number and case), we introduce two-character positional tag: one character for number (S=singular, P=plural) and one for case (1-7). That is slightly different from the approach of Kann and Schütze [2016], who needed to encode more information to the tag since their system did not aim specifically on nouns and inflection according to number and case only, and therefore used variable number of features. In addition, we introduce a special token to separate the lemma from the tag in the input. We think that this could be beneficial since it allows the model to distinguish between the lemma and the morphological tags.

We use individual characters of the form as output, as proposed by Kann and Schütze [2016]. Also we use the start-of-sequence and end-of-sequence tokens both in the input and the output (it is included by default in OpenNMT).

In this way we define a baseline data representation: the input consists of characters of the input lemma, followed by the separator token, followed by the tag, where each feature has its own token. The output consists of characters of the desired form only. In Table 4.2 we can see an example for Czech word *jablko* (an apple) with the form in instrumental of plural.

Furthermore, we propose some experiments to compare the performance of systems using different data representation, namely: tag as one token or as two separate tokens; using the separator token or not; tag at the beginning of the input or at the end; reversed characters in the input lemma and the output form or not. We report the results of comparison of different data representations in Section 5.4.3.

We could also discuss what to include in the training data. The dataset contains non-existent-form tokens. If we do not drop the corresponding lemma-tag-form triples from the data we use for training, the network learns to predict the non-existent forms too. If our aim was deciding whether a given desired form exists and generating it only if it does, it would be reasonable to keep the entries in the training data. Yet our aim is to predict the form, regardless of how probable its existence is. Therefore it may be beneficial to

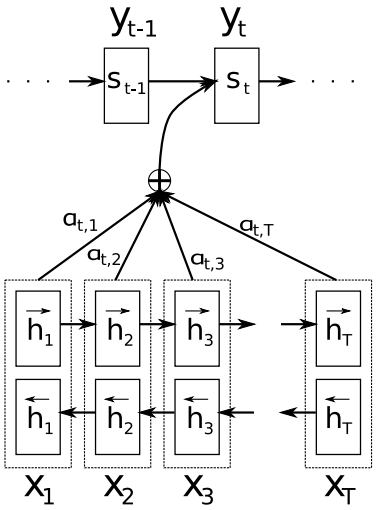

Figure 4.1 RNN seq2seq architecture [Bahdanau et al., 2016, Figure 1]; 1-layer bi-directional encoder (at the bottom), 1-layer decoder (at the top); x_i = input character, y_i = output character, h_i = encoder hidden state, $a_{t,i}$ = attention weight, s_i = decoder hidden state

drop the corresponding entries.

The other question for discussion are lemma-tag-form duplicates. Those are present in the data because of homonyms. Although they have different paradigm tables, usually they share some forms - and for these forms, we obtain duplicate lemma-tag-form entries. The presence of the duplicates could be problematic, because they add importance to the specific forms. Nevertheless, it is difficult to decide whether to include or exclude them by theoretical means.

We investigated the impact of excluding the non-existent forms or lemma-tag-form duplicates from the training data to the performance of models. The experiments are briefly described in Section 5.4.4.

4.3.3 Architecture and hyperparameters

RNN encoder-decoder

Our RNN model is character-level encoder-decoder with attention. See Figure 4.1 for a schema of the architecture. We adapt the architecture from Kann and Schütze [2016] (we denote the approach of these authors as "KS-16" in this text). We use LSTM units [Hochreiter and Schmidhuber, 1997] as the type of RNN both in the encoder and the decoder instead of GRU [Chung et al., 2014], because it has been shown that LSTM performs better than

GRU on larger datasets with shorter sequences [Yang et al., 2020].

Unlike KS-16, we employ Luong attention [Luong et al., 2015] instead of Bahdanau attention [Bahdanau et al., 2016], since it is used by default in OpenNMT. Unlike KS-16, we try not only the bidirectional encoder but also the standard one-directional. Since our training dataset is much larger than the SIGMORPHON's 2016 dataset used by KS-16, we examine extending the capacity of the network by increasing the number of hidden layers both in the encoder and decoder (KS-16 used only one layer) and increasing the number of the RNN units in each layer (KS-16 used 100 hidden units in each layer). We also experiment with the size of character and tag embeddings. Since the input and the output sequence share most of the vocabulary, we experiment with shared embeddings.

Transformer

Our Transformer model architecture is adapted from Wu et al. [2021]. It is also an encoder-decoder architecture, yet the layers are Transformer layers. Unlike the RNN-based model, it does not employ recurrent connections. It relies on self-attention which captures the dependencies between different characters in the sequence.A high-level schema of the Transformer model on the task of morphological inflection is in Figure 4.2[3].

The most important hyperparameters are the number of layers (in Figure 4.2, there are 6 layers both in the encoder and the decoder), size of the hidden layers, the number of self-attention heads, the size of the feed-forward layer, and embedding size, all affecting model's capacity. To allow regularization, we make use of dropout and attention dropout.

Unlike Wu et al. [2021], we use standard positional encoding [Vaswani et al., 2017].

For more thorough description of the Transformer architecture and its hyperparameters refer to Section 2.1.4.

Training parameters

Unlike Kann and Schütze [2016] (KS-16) we use SGD or Adam as the optimizer for training instead of Adadelta.

Since our training dataset is extremely large (approximately 5M lemma-tag-wordform triples), we need to experiment with the training parameters: the batch size and the number of training steps, which together influence the

[3]adapted from `https://jalammar.github.io/images/t/The_transformer_encoder_decoder_stack.png`

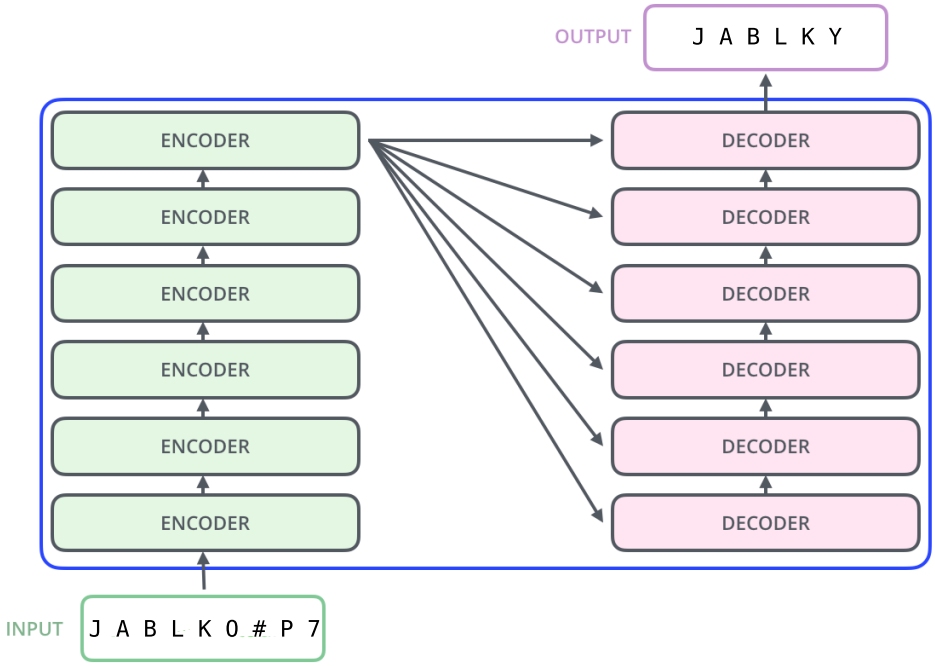

Figure 4.2 Transformer schema: 6-layer encoder-decoder on the task of morphological inflection.

number of epochs (see equation (4.1)).

$$\#\text{ epochs} = \frac{(\text{batch size}) \cdot (\text{train steps})}{\text{train size}} \tag{4.1}$$

For example, with batch size 1000 we need approx. 5000 training steps to perform one epoch of training, and with batch size 20 as used by KS-16, we would need 250k training steps to complete one epoch.

Consequently, we experiment with larger batch size and still with lower number of epochs, compared to KS-16 and Wu et al. [2021]. See Section 5.4 for reports of experimenting with training parameters.

Chapter 5

Experiments and results

In this chapter we describe the experiments we performed and the obtained results on our datasets.

We start by describing the metrics used for evaluation and briefly reminding which data are used for the experiments. Then we evaluate all the systems on the development data. First we describe the evaluation of baseline performance. Then we report experiments with our retrograde model and our seq2seq models and the process of choosing the best paramaters (configuration). Finally we report and compare results of all the systems on the test and test-oov datasets. Moreover we evaluate our best system on SIGMORPHON's 2022 dataset for several languages and compare the performance with some other systems.

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

5.1 Evaluation setup

5.1.1 Metrics

To evaluate the performance of our models, we use two metrics, both utilized in SIGMORPHON's 2017 Shared task [Cotterell et al., 2017]:

1. form accuracy (FA): is the predicted form (paradigm cell) correct? (5.1)
2. full-paradigm accuracy (FPA): is the complete paradigm table correct? (5.2)

$$\text{FA} = \frac{\#(\text{correctly predicted forms})}{\#(\text{all existent gold forms})} \tag{5.1}$$

$$\text{FPA} = \frac{\#(\text{correctly predicted full paradigm tables})}{\#(\text{all lemmata})} \tag{5.2}$$

We always report both the accuracies in percent (but without the % sign).

Unlike Cotterell et al. [2017], we do not use any metric to reflect the Levenshtein distance of the predicted form from the gold target form. We consider the exact match accuracy: the predicted form is either correct or not.

Full-paradigm accuracy is computed per lemma, while the form accuracy is computed per form. The form accuracy of a system is always higher (or equal) to the full-paradigm accuracy.

Since all our datasets contain special tokens for non-existent forms, and the test-oov dataset contains multiple possible forms for some paradigm cells, we need to deal with it properly.

Therefore we define our evaluation in the following way:

- A predicted form is ignored if it is marked as non-existent in the gold data.
- A predicted form is considered correct if it is not ignored and it is equal to the target gold form. If multiple correct answers are possible, we accept any string from the correct set.
- A predicted form is considered incorrect otherwise (if it is not ignored and it is not equal to any of the possible target forms).
- A lemma (paradigm table) is considered incorrect if it contains any incorrect form, otherwise it is considered correct.
- The full-paradigm accuracy is computed over all lemmata and the form accuracy is computed over the existent forms (forms that are not marked as non-existent in the gold data).

In simple terms, we do not care what the model produces for the paradigm cells marked as non-existent (we do not consider the produced form neither correct nor incorrect).

If our aim were to produce a model that can also recognize whether a target form exists or not, it would be sensible to evaluate also the predictions on the non-existent forms.

lemma	tag (case question)	predicted form	(gold form)
	S4 (vidím)	Dunaje	(Dunaj)
Dunaj	P1 (ti/ty/ta)	Dunaji	(Dunaje)
	P5 (volám)	Dunaji	(Dunaje)
cín	S2 (bez)	cína	(cínu)

Table 5.1 Inspection of results: example of incorrectly inflected forms.

5.1.2 Logs: inspection of results

To be able to focus on specific errors produced by the model, we log the evaluated predictions.

For every lemma that contains some incorrectly predicted forms we log the lemma together with the incorrect predictions. Each form is marked by the tag and the case question describing the corresponding paradigm cell. The gold target form is included in parentheses.

In Table 5.1 we can see an example from the log file for two lemmata: *Dunaj* (the river Donau) and *cín* (tin). For lemma *Dunaj* there are three incorrect forms (one in singular, two in plural), meaning that the rest 11 forms were correct. For the lemma *cín* there is only one incorrect form.

5.1.3 Data used

We use the train set (360k lemmata) for training the models (extracting the paradigms in the retrograde model). For development of the models, adjusting the hyperparameters and choosing the best configuration (Sections 5.2, 5.3, 5.4) we use the development set (44k lemmata). The test set and test-oov set are only used for the final evaluation and comparison of all models (Section 5.5 exclusively). For the comparison of our best model on a standard dataset we use the SIGMORPHON's 2022 dataset for all development languages that included large training dataset (18 languages, excluding Czech but including Slovak).

5.1.4 Upper bound on accuracy

Since we included homonyms in the datasets, there are some lemma-tag pairs that are equal but have different target form. Therefore it is not possible to achieve 100% accuracy (at least for any deterministic model that for a given lemma and tag predicts always the same form).

Accordingly there is an upper bound on accuracy for every dataset. The upper bound for the full-paradigm accuracy can be computed simply: it is

the number of unique lemmata divided by the total count of lemmata (if there are two lemmata with different paradigm tables, only one of them can be predicted correctly). The upper bound for the form accuracy is always higher than the upper bound for the full-paradigm accuracy.

To reflect this property of our datasets, we include the upper bound on the full-paradigm accuracy to the evaluation and comparison (Oracle system).

5.2 Baselines

The baselines are described in Section 4.1. Here we discuss the technical aspects of running some of the systems on our data, and evaluate all baselines on development data.

5.2.1 Non-neural SIGMORPHON

We start by running the non-neural baseline on 2022 SIGMORPHON data [Kodner et al., 2022], evaluating the predictions using their script for evaluation and checking that result are identical to those reported by the organizers. Our evaluation method aims beside other at evaluation of the full-paradigm accuracy, and as such it relies on having whole paradigm tables in the data, and thus is not suitable for evaluation of predictions on SIGMORPHON data (which rather consists of individual forms).

After checking that the system has the same results as the system actually reported, we run it on our data converted to the SIGMORPHON data format. Although the authors claim that the system was built for speed of application [Cotterell et al., 2017], we observe exceptionally slow performance on our large datasets (360k training lemmata, 44k development lemmata): extraction of the rules from training data took approximately 2 hours, and prediction of all forms for the development lemmata took almost 5 days when running on one CPU. We adjusted the code by removing the evaluation parts, which included counting of accuracies for unseen lemmata and unseen features separately, leaving there the production of predicted forms only. This modification sped it up and the prediction of forms for development data took less than 2 days (the speed is approx. 4 predicted forms per second). That is still quite slow.

5.2.2 Neural SIGMORPHON

As with the non-neural baseline, we replicate the experiments on SIGMORPHON data with the neural one too. Nevertheless, although the default setting of the baseline seems to be consistent with its description [Pimentel

transformer	batch	steps	epochs	FA	FPA
inp-inv	400	20k	2	95.33	87.71
inp-inv	800	20k	4	95.54	88.47
inp-inv	1200	20k	5	95.60	88.22
inp-inv	2000	20k	8	95.71	89.02
inp-inv	400	100k	8	95.72	89.17
inp-inv	400	150k	12	95.82	89.62
inp-inv	800	150k	24	95.93	89.89
vanilla	400	20k	2	95.31	87.75
vanilla	800	20k	4	95.54	88.22
vanilla	1200	20k	5	95.63	88.34
vanilla	2000	20k	8	95.74	88.93
vanilla	400	100k	8	95.73	89.25
vanilla	400	150k	12	95.84	89.46
vanilla	800	150k	24	**95.96**	**89.98**

Table 5.2 SIGMORPHON neural baselines - development evaluation. The baselines trained with different batch size and for different number of maximal training steps. The number of epochs is equal to the number of checkpoints that are available for model selection. inp-inv = input-invariant transformer [Wu et al., 2021], vanilla = standard vanilla transformer [Vaswani et al., 2017]. FA = form accuracy, FPA = full-paradigm accuracy, both in %.

et al., 2021; Wu et al., 2021], we obtained slightly different results of the input-invariant transformer on SIGMORPHON's 2022 data (both in small and large data conditions). The results are not consistently worse nor better. In most of the languages the difference in accuracy is up to 1 percentage point in the large data condition and up to 3 points in the small data condition, yet in some languages the accuracy of our run of the experiment is improved by up to 41% over the reported results (in both data conditions). Nonetheless, since the results are either better than reported or only slightly worse, we think that we can still use the systems for comparison, yet keeping in mind that the setting may be to some degree different from the setting of baselines reported in the shared task.

A more significant issue is the impossibility of using model selection with their setting of hyperparameters together with our dataset, as already discussed in Section 4.1.3.

To allow the usage of development data to choose the best model amongst several checkpoints, we apply at least some experiments with increased batch size and number of training steps. We report the results of these experiments in Table 5.2.

baseline	FA	FPA
Copy-base	22.53	1.53
Sklonuj.cz	88.72	74.19
SIG non-neural	94.60	87.95
SIG inp-inv	95.33	87.71
SIG vanilla	95.31	87.75
SIG inp-inv long-train	95.93	89.89
SIG vanilla long-train	**95.96**	**89.98**
oracle	$>$ 99.18	99.18

Table 5.3 Baselines - development evaluation. SIG non-neural = SIGMORPHON non-neural baseline, SIG inp-inv = SIGMORPHON 2021 neural baseline - input-invariant transformer with default setting, SIG vanilla = SIGMORPHON 2021 neural baseline - vanilla transformer with default setting, long-train = the same with longer training (24 epochs).

The baselines without any change in training parameters perform relatively well, yet the number of epochs is too low. We can see that both accuracies improve when increasing the batch size and more significantly when increasing the number of train steps. They achieve the best results when training for 150k steps with batch size 800, which is the largest training we tried. It seems that it could be beneficial to try to extend the training more. Nonetheless, with increasing batch size and training steps increases also the need of computational resource and training time, and therefore we leave it here for future research.

For final evaluation we choose the baselines with original setting (batch size 400, trained for 20k steps) and the best models with longer training(batch size 800, trained for 150k steps).

5.2.3 Baseline results

We report the results of all the baselines on the development dataset in Table 5.3.

The performance of the copy baseline reflects the properties of the dataset: in all paradigm tables together, 22.5% of the forms are equal to the lemma, and 1.5% of lemmata are inflexible (all forms are equal to the base form).

The rule-based system skloňuj.cz performs poorly compared to SIGMORPHON baselines, in both accuracies. SIGMORPHON non-neural baseline outperforms the neural baselines (with default setting) in the full-paradigm accuracy, but achieves worse result in the form accuracy. The best performing baseline is SIGMORPHON vanilla Transformer with long training, achiev-

Model	trainsize	combine	FA	FPA
Copy-baseline	-	-	22.53	1.53
Retrograde	1	1	22.63	1.61
Retrograde	100	64	83.17	72.95
Retrograde	800	512	90.96	81.35
Retrograde	80k	65536	94.01	87.00
Retrograde	250k	131072	94.52	88.08
Retrograde	360k	131072	**94.67**	**88.43**

Table 5.4 Retrograde model - partial development results: combine=max number of paradigms for combination, trainsize = number of training lemmata.

ing almost 96% in the form accuracy and almost 90% in the full-paradigm accuracy.

However, we can see that there is still place for improvement, since the gap between the best achieved result and the best possible result achievable on the dataset by a deterministic system (reflected by the entry of oracle system) is almost 10% in the full-paradigm accuracy and more than 3% in the form accuracy.

5.3 Retrograde model

In our experiments, we used the training dataset as the known lemmata with full paradigm tables. We built the retrograde model and evaluated the performance on development data.

In total, we performed 319 experiments, changing the number of training paradigms used to build the model and changing the maximal number of paradigms allowed to combine. The latter can be applied when there are multiple training lemmata that have the same longest common suffix with the given lemma, and therefore there is no single lemma to be used as the paradigm for inflection. In such case it is possible to restrict the maximal number of lemmata to combine.

We investigate performance for 27 several different sizes of training set (from a single training lemma up to the full training set containing 360k lemmata). For the maximal number of combinations we try the powers of 2, not examining numbers of combinations that are larger than the training size.

We observed that for a fixed number of training paradigms, the best performance was achieved when allowing as much combinations as applicable. Nevertheless, for some training set sizes there were exceptions, where a higher number of combinations led to a slight drop in accuracy, which we guess

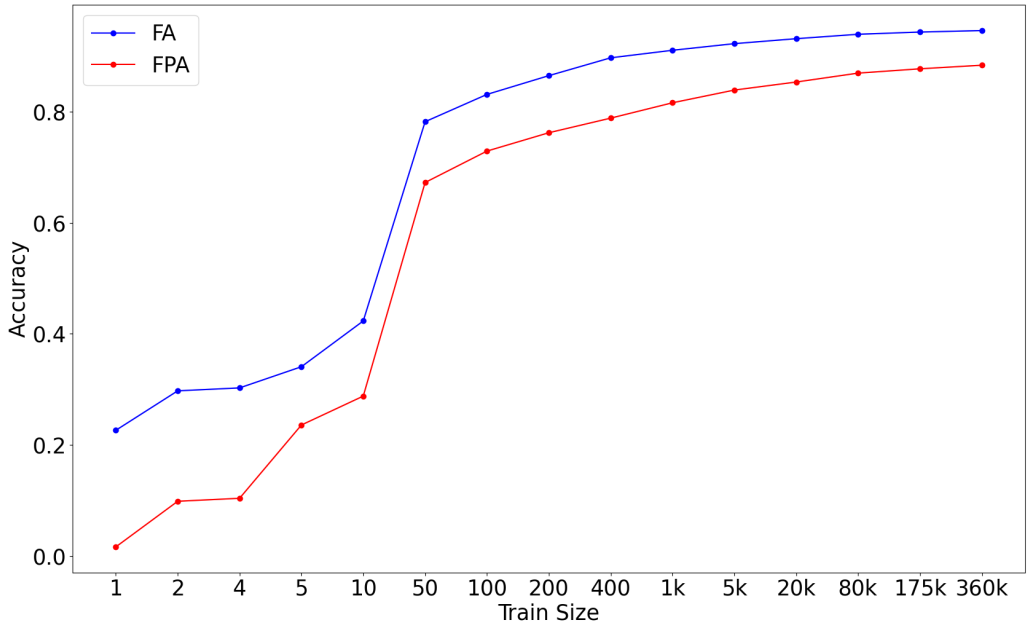

Figure 5.1 Retrograde model development evaluation with different sizes of training dataset. Train size = count of lemmata in the training dataset. The horizontal axis corresponds to the experiments that we carried out, it is neither linear nor logarithmic.

was caused by the specifics of the development dataset. For each size of training set we therefore report the results for the largest number of allowed combinations.

In Table 5.4 we present selected results for some training set sizes. Figure 5.1 plots the relation between the size of training dataset and accuracy on development dataset. Full results are reported in Appendix A.

Generally, the more training data we have, the better results we get.

It is interesting to note that while the best achieved form accuracy is 94.7% and the best achieved full-paradigm accuracy is 88.4%, even with 800 training paradigms we already surpass both 90% in form accuracy and 80% in full-paradigm accuracy. Moreover, already with 80k paradigms we achieve 94% form accuracy and 87% full-paradigm accuracy. In addition, even with train size 1 (that is, when randomly choosing one training example as the paradigm for all lemmata in development set) we surpassed the accuracy achieved by the copy baseline (yet this can be caused by having a bit of luck when choosing the one random example).

From the results we can see that if there are no issues with used disk space nor with speed of prediction, the best setting is to allow as many training paradigms as possible and also make no restrictions about the maximal

number of combinations.

The model we choose for final evaluation is therefore the one trained on the full training set (360k lemmata) and with maximal number of combinations 2^{17}.

5.4 Seq2seq model(s)

In this section we report the workflow of our experiments with RNN-based encoder-decoder and with the Transformer model. We discuss the important hyperparameters and their values we found most appropriate. All results we report here are on the development set.

All experiments were conducted using the OpenNMT library. It uses token-wise cross-entropy loss as the training loss for seq2seq models.

Both for the RNN architecture and the Transformer we choose the best trained model for final evaluation.

All the models where trained and evaluated on one GPU. To allow reproducibility of the experiments, we used a single predefined random seed for all of them.

In general, it seems that overfitting is not an issue with our datasets, probably because the training set is extremely large.

5.4.1 RNNs

We investigate using stochastic gradient descent (SGD) and Adam optimizers for training. For both of them we use the learning rate recommended by OpenNMT: 1.0 for SGD and 0.001 for Adam. We always use the whole sequences (not individual characters) for batching. Unless we explicitly state, we use the default setting from OpenNMT[1]. In all experiments we use LSTM as the RNN type both in the encoder and the decoder, and we use 1000 warmup steps in training.

We limit the vocabulary size for the embeddings to 50k distinct "words" (characters and tags in our task), which has no effect since the number of distinct characters and tags in our dataset is less than 200. We limit the length of the source and the target sequences to 150 characters. Nor this has any effect because the sequences in our dataset are short. When using shared embeddings for the encoder and decoder, we use also shared vocabulary.

Training for high number of epochs with small batch size takes too much time. Due to limited amount of computational resources we decide to increase the number of epochs by using larger batch sizes (up to 512).

[1] `OpenNMT-py v3.0.4`

model	description	steps	epochs	FA	FPA
LSTM-2	emb=32,lay=2,batch=20	20k	0.08	74.40	48.34
LSTM-3	lay=2,emb=500,batch=64	20k	0.25	94.00	80.28
LSTM-13	emb=500	20k	1	95.13	85.62
LSTM-14	emb=500,batch=512	20k	2	94.92	84.98
LSTM-17		40k	2	95.16	86.58
LSTM-18	emb=256	40k	2	95.04	85.32
LSTM-19	emb=64	40k	2	95.26	86.54
LSTM-20		60k	3	95.62	88.71
LSTM-21	SE	60k	3	95.67	88.63
LSTM-22	emb=96,SE	60k	3	95.68	88.70
LSTM-23a	SE	120k	6	95.94	89.66
LSTM-23b	SE	240k	12	**95.95**	**89.70**

Table 5.5 LSTM SGD-trained dev evaluation. All systems: size of hidden layers 500. Unless stated otherwise, batch size is 256 (batch), number of hidden layers 4 (lay), word-vec size for embeddings 128 (emb). SE=shared embeddings between the encoder and the decoder (by default not shared)

It is important to note that we did not perform exhaustive parameter tuning. Therefore all conclusions about how the hyperparameters affect the model performance are rather our tentative hypotheses than definite claims.

Training with SGD

We report the results of experiments of training RNN-based encoder-decoder with SGD in Table 5.5. In the following paragraphs, we refer to the table by model name.

LSTM-2, 3, 13 With SGD, we verified that training for less than 1 epoch (omitting a random part of the training set) leads to performance drop in both accuracies, more significantly in the full-paradigm accuracy.

LSTM-13, 14 It seems that the longer we train the models, the better, at least up to the extent we were able to test. Yet some experiments indicate that this should not be led by unlimitedly increasing the batch size, but rather by increasing the number of training steps. Increasing the batch size too much can lead to performance drop.

LSTM-22, 23a, 23b Increasing the number of epochs by increasing the number of steps leads to an improvement.

LSTM-13 up to 23 In all relevant experiments we run with SGD (those that were trained for at least one epoch), the capacity of the model remained almost the same, with minor changes in the setting of embeddings (embedding size, shared embedding between encoder and decoder). The number of layers both in the encoder and the decoder is set to 4 and their size is 500. We use the standard one-directional RNN both in the encoder and in the decoder. We use Luong attention [Luong et al., 2015] (named "general" in OpenNMT).

LSTM-17, 18, 19 We tried several different embedding sizes (64, 96, 128, 256). It seems that embedding size 256 is too large - a model with embedding size 256 performed worse than the same model with embedding size 128, which performed comparably to the same model with embedding size 64.

LSTM-20, 21 Sharing the embeddings between encoder and decoder seems to have only minor effect on the model's performance. The experiment we executed showed a slight improvement in the form accuracy and a slight drop in the full-paradigm accuracy when using shared embeddings.

LSTM-23b The best performing LSTM model trained with SGD has shared embeddings of size 128, 4 encoder and 4 decoder layers of size 500 and was trained for 240k steps with batch size 256 (approx. 12 epochs). On the development set it achieved form accuracy 95.95% and full-paradigm accuracy 89.7%.

Training with Adam

Next, we try training the models with Adam [Kingma and Ba, 2017] (learning rate 0.001) instead of SGD. We extensively reduce the capacity to 1 layer of size 100 and embedding size 64. Except for one experiment performed to compare different attention types, we use Luong attention [Luong et al., 2015]. In all experiments we use shared embeddings, bi-directional RNN in the encoder, and we train with 4k warmup steps. We report the results in Table 5.6.

LSTM-26, 27 We start by training with batch size 20 (used by Kann and Schütze [2016]) for 1 epoch and 4 epochs, without any improvement in longer training.

LSTM-28 Increasing the capacity by setting the hidden layer size to 150 units, we obtain an improvement in both accuracies.

model	description	ep	FA	FPA
LSTM-26		1	94.82	85.64
LSTM-27	steps=1040k	4	94.82	85.64
LSTM-28	hid=150	1	94.94	86.05
LSTM-29	hid=150,attn=Bahdanau	1	94.93	85.95
LSTM-30	hid=200	1	95.14	86.76
LSTM-31	hid=250	1	95.13	86.73
LSTM-32	hid=250,steps=520k	1	94.98	86.26
LSTM-33	lay=2	1	94.83	85.96
LSTM-34	lay=2,steps=1080k	4	94.83	85.96
LSTM-35	lay=2,hid=150	1	95.12	86.82
LSTM-36	lay=2,batch=128,hid=150	7	95.67	88.41
LSTM-37	lay=2,batch=256,hid=150	13	95.83	89.11
LSTM-38	lay=2,batch=32,hid=150,steps=2M	13	95.31	87.39
LSTM-39	lay=2,batch=400,hid=150	21	95.87	89.04
LSTM-40	lay=2,batch=256,hid=200	13	95.97	89.44
LSTM-41	lay=2,batch=256,hid=250	13	95.94	**89.46**
LSTM-44	lay=2,batch=256,emb=128,hid=200	13	**95.98**	**89.46**
LSTM-45	lay=3,batch=256,emb=128,hid=200	13	95.96	89.26
LSTM-46	lay=3,batch=256	13	95.62	88.19
LSTM-47	lay=2,batch=256,emb=16	13	95.16	86.70

Table 5.6 LSTM Adam-trained dev evaluation. ep=epochs. All systems: learn rate=0.001, brnn encoder, warmup4k, shared embs. Unless stated otherwise, batch size is 20 (`batch`), steps 260k (`steps`), number of hidden layers 1 (`lay`), hidden layer size 100 (`hid`), word-vec size for embeddings 64 (`emb`), Luong attention (`attn`).

LSTM-28, 29 We compare utilizing Luong attention [Luong et al., 2015] ("general" in OpenNMT, default in OpenNMT) and Bahdanau attention [Bahdanau et al., 2016] ("mlp" in OpenNMT, used by Kann and Schütze [2016]) and we see that a model with Bahdanau attention performs slightly worse than a model with Luong attention.

LSTM-26, 28, 30, 31 We try four different values for the hidden layer size (100, 150, 200, 250), obtaining the best results with the value 200.

LSTM-31, 32 With hidden size 250 we compare training the model with two different (batch size, train steps) pairs, both corresponding to one epoch, obtaining better results with batch size 20 and 260k train steps than with batch size 10 and 520k train steps.

LSTM-26, 33 Using two layers of size 100 instead of one layer of the same size leads to a slight improvement in both accuracies.

LSTM-33, 34 However, even in this case, training for more steps does not lead to almost any further improvement.

LSTM-35, 31 Nevertheless, increasing the hidden size to 150 leads to improvement and we surpass the best model with 1 layer in the full paradigm accuracy.

LSTM-36, 37, 38 We achieve more significant improvement by increasing the batch size up to 256 (and therefore increasing the number of epochs up to 13). Moreover we show that a model trained for more steps with smaller batch size (corresponding to approx. the same number of epochs) performs significantly worse, particularly in the full-paradigm accuracy. This does not correspond to the general trend of using smaller batch size for RNN-based encoder-decoder architectures for morphological inflection [Kann and Schütze, 2016].

LSTM-37, 39 Further increasing the batch size to 400 leads to a slight improvement in the form accuracy, yet the full-paradigm accuracy decreases a bit.

We continue performing experiments with batch size 256 and 260k training steps (approx. 13 epochs).

model	optimizer	description	epochs	FA	FPA
LSTM-23b	SGD	see Table 5.5	12	95.95	**89.70**
LSTM-44	Adam	see Table 5.6	13	**95.98**	89.46

Table 5.7 LSTM: comparison of best systems trained with SGD (learning rate 1.0) and with Adam (learning rate 0.001).

LSTM-47 Reducing the embedding size to 16 and the hidden size to 100 leads to a drop in both accuracies, yet not exceptional.

LSTM-41 On the other hand, increasing the hidden layer size to 250 leads to our best result in full-paradigm accuracy with Adam training, 89.46%.

LSTM-40, 44 With hidden layer size 200 and embedding size 64 we achieve better result in the form accuracy, and after increasing the embedding size to 128 we obtain the best performing LSTM model trained with Adam: it achieves 95.98% in the form accuracy and 89.46% in the full-paradigm accuracy.

LSTM-45, 46 The models with 3 hidden layers perform slightly worse.

Best LSTM model

We showed that our models perform better when trained with larger batch size (256) and with less training steps, compared to training the same number of epochs with smaller batch size and more training steps.

Regarding the form accuracy, we achieved the best results on the development set with the Adam-trained model **LSTM-44**: 2 layers of size 200 both in the encoder and decoder, with shared embeddings of size 128, with bi-directional LSTM in the encoder and training with batch size 256 for 260k steps (approx. 13 epochs) with 4k warmup steps. With this model we achieved almost 96% in the form accuracy and almost 89.5% in the full-paradigm accuracy. The best SGD-trained model (LSTM-23b) achieved comparable accuracies (slightly lower FA, slightly higher FPA, see Table 5.7 for comparison). Deciding based on the form accuracy, we choose the Adam-trained model LSTM-44 for final evaluation.

Technical obstacles

During our experimental work, we encountered several issues. We mention them so that other researchers can learn from our mistakes.

Initially, we draw conclusions about some hyperparameters without knowing that we are not training at the whole dataset. An important reminder: with the batch size 20 and the size of the train set 5M, we need 250k training steps to perform one epoch. When training for less steps, we omit a random part of the train set.

Initially, training took extreme amounts of time because we did not set the validation batch size parameter, which is by default set to 32 only. With our large development set it led to the fact that during the training we spent more time by evaluating the model on the development set than by performing the training steps themselves.

5.4.2 Transformers

In addition to experiments with LSTM-based encoder-decoder, we conducted several experiments with Transformers. We investigated three different settings. One is adapted from an OpenNMT tutorial, the other one is adapted from Wu et al. [2021] and the last one is a Transformer with extremely low capacity.

Unless explicitely stated, we use the gradient accumulation with count of 4 batch steps, which means that the gradient is accumulated for 4 batches before performing one training step (weights update). It means that the effective batch size is 4 times larger than the batch size we set.

Mini-transformer

All experiments with mini-transformer are reported in Table 5.8. Initially it seemed like a good idea to start with a low capacity model and gradually increase it. Therefore we tried a Transformer model with extremely low capacity: only 1 layer of size 64, embedding of size 64, 1 attention head, feed-forward of size 128. We train it with dropout and attention dropout set to 0.2 and with batch size 32 (effective batch size 128 due to accumulation of gradient).

TRM-4, 5, 6 We report results for training the model for 51, 128 and 305 epochs. The model is steadily improving, yet the model trained for the longest time does not even get 70% in the form accuracy, although being trained more than 3 days.

TRM-8 Due to the long training time we rather try increasing the batch size to 512 (effective batch size to 2048), yet it is trained for more epochs than the previous models, but achieves worse results.

model	description	steps	epochs	FA	FPA
TRM-4		2M	51	31.13	0.28
TRM-5		4M	102	50.52	0.70
TRM-6		12M	305	**67.94**	**35.80**
TRM-8	eff_batch=2048	2M	813	59.71	19.76
TRM-7a	heads=2	2M	51	27.97	0.25
TRM-7b	heads=2	4M	102	45.32	0.39

Table 5.8 Transformer mini dev evaluation. All systems: 1 layer of size 64, embedding of size 64, feed-forward layer of size 128, dropout=0.2, attention dropout=0.2. Unless stated otherwise, batch size=32 (effective batch size 128 (`eff_batch`)), used 1 attention head (`head`).

TRM-7a, 7b We also try a model with 2 attention heads, but we achieve worse results than with 1 head.

Consequently we decide not to perform more experiments with the small-capacity Transformer. Nonetheless we think that in future research it could be beneficial to experiment with the small-capacity Transformer, gradually increasing the capacity and seeing when it starts to achieve acceptable results.

Transformer from Wu et al. [2021]

All corresponding experiments are reported in Table 5.9.

In the next experiments with the Transformer architecture, we adapt the hyperparameters from Wu et al. [2021]: The model has 4 layers of size 256 both in the encoder and in the decoder, 4 attention heads, the feed-forward network has size 1024. It is trained with Adam with inverse square root learning rate decay, starting at learning rate 0.001 with beta parameter 0.98. We use layer normalization and dropout 0.3, attention dropout 0.1 and label smoothing 0.1.

TRM-12, 13a We train it with batch size 400 (setting accumulation of gradient to just one step, effectively having batch size 400) for 20k steps (1.6 epochs). Moreover we train the same model with batch size 800 for 10k steps and we obtain better results with the smaller batch size.

TRM-13b, 15 After increasing the number of training steps up to 300k the performance increases.

TRM-9a-c, 10a-c Using larger effective batch size (1600 or 16384) and training for more steps (up to 500k) together lead to further improvement,

model	description	steps	epochs	FA	FPA
TRM-9a	accum=4	20k	6	88.96	65.25
TRM-9b	accum=4	200k	63	94.10	83.22
TRM-9c	accum=4	400k	127	94.53	85.17
TRM-10a	accum=4,batch=4096	100k	325	93.48	80.54
TRM-10b	accum=4,batch=4096	300k	975	94.33	84.41
TRM-10c	accum=4,batch=4096	500k	1625	**94.60**	**85.54**
TRM-12		20k	2	87.58	62.88
TRM-13a	batch=800	10k	2	86.52	61.81
TRM-13b		200k	16	94.02	82.66
TRM-15		300k	24	94.31	83.73

Table 5.9 Transformer from [Wu et al., 2021] dev evaluation. All: The model has 4 hidden layers of size 256 both in the encoder and in the decoder, 4 attention heads, the feed-forward network has size 1024. It is trained with Adam with inverse square root learning rate decay, starting at learning rate 0.001 with beta parameter 0.98. We use layer normalization and dropout 0.3, attention dropout 0.1 and label smoothing 0.1. Unless stated otherwise, we use batch size 400 (`batch`), and accum_count 1 (`accum`). Effective batch size can be computed as `eff_batch=batch*accum`.

yet not surpassing the best performing LSTM model we have.

Transformer from tutorial

Further, we try the Transformer setup from an OpenNMT tutorial[2] (focused on machine translation). All experiments with such Transformer are reported in Table 5.10.

TRM-2 Training the model as suggested by the OpenNMT tutorial, only changing the number of training steps from 3k to 100k, the number of warmup steps from 1k to 4k, and omitting early-stopping, leads to surprisingly good results.

The model has extremely high capacity: embeddings of size 512, 6 layers of size 512, 8 attention heads and feed-forward of size 2048. It is trained with Adam with "noam" decay, starting at learning rate 2, with beta parameter 0.998. For regularization it uses layer normalization and dropout 0.1, attention dropout 0.1 and label smoothing 0.1. It is trained with a very large batch size (4096) and with accumulation count 4, yet the batches are built per tokens (not per whole sequences) and therefore we cannot directly compare the batch

[2] `https://github.com/ymoslem/OpenNMT-Tutorial/blob/main/2-NMT-Training.ipynb`

model	steps	eff batch	epochs	FA	FPA
TRM-2	100k	-	-	94.80	85.67
TRM-3	100k	256	5	95.55	88.41
TRM-4	200k	128	5	95.39	88.05
TRM-11	40k	4096	32	**96.08**	**90.20**

Table 5.10 Transformer from tutorial - dev evaluation. TRM-2 model has dropouts (dropout and attention dropout) set to 0.1 and hidden layer size and word-vec embedding size to 512. Other models have dropouts 0.2, hid=256, word-vec=256. Due to batching per tokens in TRM-2 it is not possible to determine neither the effective batch size (eff batch), nor number of epochs. TRM-2,3,4 were evaluated on a smaller subset of development dataset (10k lemmata randomly sampled from the whole 44k).

size here and in other experiments, neither compute the number of epochs for this experiment.

TRM-3, 4 In the next experiments we build batches by sentences, not by tokens, increase dropout and attention dropout from 0.1 to 0.2, decrease the hidden size and word-vec (embedding) size from 512 to 256. We conduct two different experiments with the same number of epochs but changing the number of training steps and the batch size. We show that in this setting the model trained with larger batch for less training steps performs better than the other one.

TRM-11 Therefore we execute another experiment with even larger batch size (1024, effective batch size 4096). After training for 40k steps (32.5 epochs) we outperform all the models we had, surpassing 96% in the form accuracy and 90% in the full-paradigm accuracy on the development set. It is our overall best performing model.

Best Transformer model

The best performing Transformer model we trained is the **TRM-11** with architecture adapted from the OpenNMT tutorial, trained with large batch size (effective batch size 4096). Since the Transformer is the current state-of-the-art in the task of morphological inflection and since we outperformed the thoroughly tuned LSTM-based models by conducting just a small number of experiments, changing the recommended setting just a little, we think it would be beneficial to execute more experiments to achieve even better results. Nonetheless, we did not have enough computational resources and time to do it and we leave it to future research.

repre	src sequence	tgt sequence	FA	FPA
Base	j a b l k o # S 7	j a b l k e m	**95.97**	**89.44**
TagBeg	S 7 # j a b l k o	j a b l k e m	95.84	88.79
TagJoin	j a b l k o # S7	j a b l k e m	**95.97**	89.36
NoSep	j a b l k o S 7	j a b l k e m	95.90	89.25
Reverse	o k l b a j # S 7	m e k l b a j	95.85	89.21
Complex	S7 j a b l k o	j a b l k e m	95.96	89.21

Table 5.11 Comparison of different data representations. Measured performance of model LSTM-40 on dev set.

5.4.3 Optimal data representation

As we suggested in Section 4.3.2, we conduct experiments to compare using different representation of the source sequence and target sequence for seq2seq models.

We try six different representations for the source sequence (lemma and tag), while the target sequence remains the same except for the representation with reversed strings.

We convert our datasets to all formats and train the LSTM-40 setup on 6 versions of the training data and evaluate it on the corresponding versions of the development dataset. Comparison of the results, together with examples of the source and target sequence is in Table 5.11.

To our surprise, the usage of the baseline data representation outperforms all other suggested representation in both accuracies, although the difference in performance is minor. The data representation with joined tag has almost the same result as the baseline representation, suggesting that tokenizing the tag does not make much difference.

The superior performance of the baseline representation (yet with only minor difference compared to other representations) could be caused by the fact that the data representation we initially chose was indeed the best one. However, we think that the reason is rather that we tuned the model hyperparameters while using the baseline data representation and therefore the model setting is adapted to that particular representation.

Nevertheless, these experiments show that the representation we chose is suitable for our task.

5.4.4 Dropping data

As discussed in Section 4.3.2, it is unclear whether it is better to drop the lemma-tag-form duplicates from the training data or keep them, and the same

train-data	FA	FPA
original	95.97	**89.44**
no lemma-tag-form duplicates	95.99	89.29
no non-existent-form entries	**96.01**	89.39

Table 5.12 Comparison of training with or without lemma-tag-form duplicates and non-existent forms. Measured performance of model LSTM-40.

for the entries with non-existent forms.

We conduct experiments to measure the impact of dropping such entries to the model performance. We use the hyperparameters and training setup of model LSTM-40. To measure the effect of dropping the lemma-tag-form duplicates from training data, we first train it with the original training set and then with a smaller dataset where the duplicates are removed.

The comparison is in Table 5.12. We can see that omitting the duplicates or non-existent forms from the training set leads to a slight improvement in the form accuracy, but also to a slight deterioration in the full-paradigm accuracy. Above all, the gaps are really small and it seems that dropping any of the suggested entry types make almost no difference. Therefore we decide to keep both types of entries in the training data.

5.4.5 Negated lemmata

The ability of Retrograde model to inflect negated lemmata when trained on data without negations is clear, because it inflects the words based on the ending segment which does not change by adding prefix "ne-" (the prefix that forms negation in Czech). Nonetheless, the ability of doing so with seq2seq models is not so clear. If the model has never seen a lemma with prefix "ne-" during training, it is not sure what it will predict for such lemma.

To evaluate the ability of seq2seq models trained on data with no negations to inflect negated lemmata, we conduct the following experiment: create a "negated" version of the development data, evaluate one of the successful models on it and compare to evaluation on the non-negated (original) development set.

As we already mentioned, negation is formed by adding prefix "ne-" (means *not*) to the word. If we have a non-negated lemma with its whole paradigm table, we can create the paradigm table for the negated lemma by adding "ne-" to all the forms. This is how the Czech language works. However, after inspecting our original data (MorfFlex) we see that not all nouns in Czech have a negated variant, e.g. the proper nouns (names).

We decide to build the "negated" version of the development dataset by

evaluation data	FA	FPA
original (no negated lemmata)	95.94	89.46
artificial negations	95.87	89.31

Table 5.13 Performance on negated lemmata: comparison of a model trained without negated lemmata on the original development dataset and on the development dataset with artificial negations. Measured performance of model LSTM-41.

leaving the proper nouns as they are and changing all the other lemmata together with their forms by adding the prefix. The negated dataset has the same size as the original dataset. This approach creates artificial negations for some lemmata that do not have the negated variant, yet surely it creates the negated variant from the lemmata actually having them. Consequently, by evaluating a model on this artificial dataset we obtain the upper bound on the performance drop we could get when evaluating on the real negated lemmata.

The results are satisfying: the performance drop is very low, approx. 0.07% in the form accuracy and 0.14% in the full-paradigm accuracy when running on the negated data.

By this experiment we showed that our models are robust to adding the prefix "ne-" to the lemmata and therefore are capable of inflecting negated variant of lemmata when trained without them.

5.5 Final evaluation

In this section we compare all our models and baselines.

We evaluate them on two different datasets: test and test-oov. For full reminder, refer to Section 3.5. The test set has the same origin as the training set and the development set (extracted from MorfFlex [Hajič et al., 2020]). The test set has empty lemma overlap with both train and development set and therefore fulfils the OOV condition. It contains approx. 44k lemmata with full paradigm tables. The test-oov dataset is OOV in the strict meaning: it contains the true OOV words (neologisms). It is relatively small (101 lemmata with full paradigm tables).

The evaluated systems are as follows. We start with the copy baseline that returns the copy of the lemma for every paradigm cell, followed by the rule-based skloňuj.cz model, which represents the ready-to-use systems available on web. We do not evaluate MorphoDiTa [Straková et al., 2014] on our datasets because of the nature of our evaluation conditions and because it uses MorfFlex as a database for inflection: it would achieve almost 100%

Model	Form acc. (FA)		Full-paradigm acc. (FPA)	
	test	test-oov	test	test-oov
Copy	22.59	13.13	1.48	0
Sklonuj.cz	88.88	86.22	74.43	55
SIG non-neural	94.78	**89.49**	88.15	**71**
SIG vanilla	95.47	87.53	87.29	63
SIG vanilla long-train	96.17	86.51	90.15	55
Retro*	94.85	89.34	88.64	**71**
*LSTM-44**	96.16	86.95	89.80	58
*TRM-11**	**96.18**	87.24	**90.44**	61
Oracle	> 99.23	100.00	99.23	100

Table 5.14 Final evaluation. Models marked with * are our models. Models written in *italics* are neural. For further specification of the models, refer to the accompanying text, and to Chapter 4.

accuracy on our test set and 0% accuracy on the test-oov dataset (because it was extracted in such a way that the words are not contained in MorfFlex). Three SIGMORPHON baselines follow: the non-neural baseline, vanilla transformer with default setting, and vanilla transformer with longer training.

From our own models there is the best chosen retrograde model (built from the full training dataset and with no restrictions on the maximal count of combinations during prediction), the best LSTM-based encoder-decoder with attention (LSTM-44, 2 hidden layers of size 200, trained with Adam for 13 epochs with batch size 256), and the best Transformer model (TRM-11, 6 layers of size 256, 8 attention heads, trained for 32.5 epochs with large effective batch size 4096).

The oracle system reflects to the property of the dataset to be correctly predicted by a deterministic system (100% is not achievable on the test set due to presence of homonyms).

We present the results in Table 5.14.

test dataset On the test dataset, the best performing model is our TRM-11 with almost 96.2% in the form accuracy and more than 90.4% in the full-paradigm accuracy. In the form accuracy, also the SIGMORPHON vanilla transformer with long training, and our LSTM-44 model perform almost the same. All neural models surpass the non-neural models by more than 0.5% in the form accuracy, yet the SIGMORPHON vanilla with default setting is outperformed by both the SIGMORPHON non-neural baseline and our retrograde model in the full-paradigm accuracy. The best performing non-

neural model is our retrograde model with almost 94.9% in the form-accuracy and more than 88.6% in the full-paradigm accuracy. The SIGMORPHON non-neural baseline is only slightly worse with almost 94.8% in the form accuracy and almost 88.2% in the full-paradigm accuracy. The rule-based model Skloňuj.cz performs significantly worse than all other models (except the copy baseline) in both accuracies, not surpassing neither 90% in the form accuracy, nor 75% in the full-paradigm accuracy.

test-oov dataset The real-world OOV dataset seems much more challenging, especially for the neural models. The largest gap in accuracy between the test dataset and the test-oov dataset is showed by SIGMORPHON vanilla with long training, which drops almost 10% in the form accuracy and almost 35% in the full-paradigm accuracy. The best performing model is the SIGMORPHON non-neural baseline with approx. 89.5% in the form accuracy and 71% in the full-paradigm accuracy, closely followed by the retrograde model, which achieves the same result in the full-paradigm accuracy and only slightly worse result in the form accuracy. The best performing neural model is the SIGMORPHON vanilla transformer with default setting with 87.5% in the form accuracy and 63% in the full-paradigm accuracy, relatively closely followed by the TRM-11 model.

This major failure of all systems on the real-world OOV test dataset suggests possible suboptimality of our training dataset for learning paradigms of true OOV words, and indicates that further research should be carried out to investigate this issue. A possible approach is starting by inspecting the actual predictions of the individual systems on the test-oov dataset[3], trying to find an explanation of the behaviour and checking to what extent the systems make similar errors.

Release of the best models We release the LSTM-44 model in a ready-to-use Python library for inflection (for further details, see Chapter 7). The LSTM model achieves similar results as the Transformer performing best on the test dataset, and is computationally less expansive and therefore more suitable for a light-weight library. We release both the TRM-11 model and the Retrograde model in developer's repository, available for further investigation (see Section 6.2.1 for more details).

[3]Included in the attachment of the thesis. See Section 6.1.2 for specific location.

5.6 Evaluation on SIGMORPHON 2022 data

To show the performance of one of our best models (LSTM-40) on some standard dataset and to compare it to another systems, we evaluate it on SIGMORPHON 2022 data [Kodner et al., 2022][4]. Although it does not contain Czech, it seems to be a convenient dataset for comparison, because it is the only shared task that extensively evaluates the performance on unseen lemmata.

Yet it is important to keep in mind that our task is different from the shared task, and therefore the performance of a model with hyperparameters tuned on our task can generally be lower. The main differences are: (i) we aim specifically at Czech while the shared task does not include it in the dataset, (ii) we use extremely large training dataset (~5M entries) while the shared task uses relatively little training dataset even in the large data condition (2k entries), (iii) we aim explicitly at the performance on the unseen lemmata while the aim of the shared task system is much broader, including performance on unseen features and other categories, (iv) unlike the shared task, we include complete paradigm tables in the datasets, and (v) we train and evaluate on all parts of speech here, although we aimed specifically at nouns during development of our setup.

We evaluate the performance on all development languages that included the large training dataset (18 languages[5] including Slovak, which is the most similar to Czech).

To be able to run our models on the SIGMORPHON data[6], we convert it to our format by tokenizing the lemma and the word form to individual characters, add the special separator token to the end of the source sequence and then add the morphological features one by one also there. The occurrence of verbs in the data leads to discovering an issue of our data representation: our models are not able to generate multiple words (phrase) as a word form (e.g. *nebol by som vzdelávaval* as *I would not have educated* in Slovak). This is due to the character-level tokenization we perform on our own by splitting the individual characters by space. An additional space (representing the actual space between the words) is then irrelevant for training. Therefore we

[4]We do not evaluate the best model, because we developed it after performing the evaluation on SIGMORPHON data.

[5]ang = Old English, ara = Modern Standard Arabic, asm = Assamese, evn = Evenki, got = Gothic, heb = Hebrew, hun = Hungarian, kat = Georgian, khk = Khalkha Mongolian, kor = Korean, krl = Karelian, lud = Ludic, non = Old Norse, pol = Polish, poma = Pomak, slk = Slovak, tur = Turkish, vep = Veps [Kodner et al., 2022]

[6]see `https://github.com/sigmorphon/2022InflectionST/tree/main/part1/development_languages` for examples

	Submitted systems					Baselines		
Lang	CLUZH	Flexica	OSU	TüM	UBC	Neural	NonNeur	OUR
ang	**76.6**	64.4	73.7	71.9	74.1	73.4	68.7	**76.6**
ara	81.7	65.5	78.7	78.5	65.5	**81.9**	50.8	79.9
asm	83.3	75.0	75.0	**91.7**	83.3	83.3	83.3	83.3
got	92.9	41.4	**94.1**	91.7	91.7	93.5	87.6	91.7
hun	93.5	62.9	93.1	92.8	91.5	94.4	73.1	**94.5**
kat	96.7	95.7	96.7	96.7	96.7	**97.3**	96.7	96.7
khk	**94.1**	47.1	**94.1**	**94.1**	88.2	**94.1**	88.2	**94.1**
kor	**71.1**	55.4	50.6	56.6	60.2	62.7	59.0	39.8
krl	**87.5**	69.8	85.9	57.8	85.4	57.8	20.8	**87.5**
lud	87.3	92.0	92.9	93.4	88.2	**94.3**	93.4	83.5
non	85.2	77.0	85.2	80.3	**90.2**	88.5	80.3	83.6
pol	96.1	85.9	94.9	74.0	95.7	74.4	86.3	**96.7**
poma	**76.1**	54.5	70.1	69.4	73.3	74.1	47.8	74.7
slk	93.5	90.0	92.2	70.4	**95.7**	71.1	92.4	95.2
tur	93.7	57.9	**95.2**	80.2	92.9	79.4	66.7	**95.2**
vep	**71.5**	58.8	70.0	57.5	68.8	59.2	60.4	69.1
average	**86.3**	68.3	83.9	78.6	83.8	80.0	72.2	83.9

Table 5.15 SIGMORPHON 2022 comparison – Feature Overlap: A test pair's feature set is attested in training, but its lemma is novel (our setting). (The languages heb and evn are excluded because their datasets did not include any such data). Except for results of our system (OUR=LSTM-40), the table was extracted from Kodner et al. [2022, Tables 17, 18 – feature rows]. "TüM" is shortcut for "TüM Main".

introduce another special token to represent the space between words in a phrase.

After the model produces the output in our format, we convert it back to SIGMORPHON data format and evaluate it using their evaluation script[7].

The test dataset contains approx. 2k lemma-tag-form entries for each language. The evaluation script reports not only the overall accuracy, but also the accuracy on different parts of the test set based on the overlap with the training set. The one relevant to us is "feature overlap", which means that the morphological feature has been seen in the training data while the lemma is novel (that is our setting).

We train the model for 260k steps with batch size 256 (approx. 9.5k epochs). We save checkpoint each 2k steps (approx. every 73 epochs), evaluate the overall accuracy on the whole development set and choose the best performing checkpoint and evaluate it on the test set.

[7] `https://github.com/sigmorphon/2022InflectionST/tree/main/evaluation`

In Table 5.15 we present results in the feature-overlap condition. We compare the performance with the neural and non-neural baseline and with all 5 submitted systems evaluated in the feature overlap condition by the task organizers, including the UBC system [Yang et al., 2022] which achieved the highest performance accross all evaluation conditions. For description of other systems see Kodner et al. [2022].

In the feature-overlap condition, our system performs particularly well and achieves the best score in 6 out of 16 languages. Averaged over all languages, our system takes shared second place with accuracy 83.9%. We perform competitively and surpass the non-neural baseline or achieve the same result in all languages except Korean (kor) and Ludic (lud). It is also interesting that in all Slavic languages (Polish (pol), Pomak (poma) and Slovak (slk)) included in evaluation we achieve rather high score. We can see that although we focused specifically on Czech morphology while tuning our setup, the model performs particularly well when trained and evaluated on other Slavic languages.

Besides, in the both-overlap condition (both the lemma and feature set of a test example are attested in the training set, but not together in the same lemma-tag-form triple) our system achieves the best score in 8 out of 18 languages that included such condition.

For full results in all evaluation conditions, see Tables A.2, A.3 in the Appendix.

The results shows that our LSTM model is not particularly language dependent and performs well also with relatively little training data.

Chapter 6

Implementation details

The first section of this chapter describes the attachment of this work and the rest some specific details of the implementation.

6.1 Attachment structure

The attachment of this work contains two repositories. The first one, `cz-inflect`[1], is a simple library for inflection.

The second one, `cz-inflect-dev`[2], is the developer's repository. It contains all code presented in this work.

Additionally, there is one text file in the attachment, `chatGPT.txt` with the original text generated by chatGPT (which we post-edited and used in section 2.1). It is included to allow complete comparison of the original text and the post-edited version.

We describe both repositories in the following two sections.

6.1.1 Inflection library

```
cz-inflect
├── build.sh
├── example_usage.py
├── inflect.py
├── models
│   └── lstm_v0.44.pt
├── run.sh
└── requirements.txt
```

[1] `https://github.com/tomsouri/cz-inflect/releases/tag/BP_official`
[2] `https://github.com/tomsouri/cz-inflect-dev/releases/tag/BP-official`

```
├── README.md
└── LICENSE
```

The inflection library consists of a very simple repository `cz-inflect`. The script `build.sh` serves for simple creation of python virtual environment and installation of dependencies (listed in `requirements.txt`). The script `inflect.py` is the inflection script itself and can be run for interactive use by running `run.sh`. A simple example usage of the inflection system as a python library is shown in `example_usage.py`. The released OpenNMT model is stored in `models`.

6.1.2 Development repository

```
cz-inflect-dev
├── configs/
├── data
│   ├── cleaned/
│   ├── processed/
│   └── log/evaluate_models/onmt/experiments/
├── docs/
├── inflection_lib/
├── LICENSE
├── Makefile
├── README.md
├── requirements.txt
├── runjob.sh
├── scripts
│   ├── data_conversion/
│   └── data_repre/
├── setup.py
└── src/czech_inflection
    ├── config.py
    ├── data.py
    ├── datastructures/
    ├── dev_testing/
    │   ├── build_data/
    │   └── eval_models/
    │       └── main.py
    └── models/
        ├── baselines/
        ├── hardcoded/
        └── retrograde_model/
```

The directory `cz-inflect-dev` included in the attachment contains the development repository of this work. Here we briefly locate the most important parts of the repository. For simplicity, instead of `src/czech_inflection` we write only `src`.

Data processing Scripts and tools for data processing are located in several different places. Scripts for MorfFlex data processing, extracting and filtering, for providing the train-dev-test split and conversion of the data to the format for neural networks are in `src/morfflex`. Scripts for processing data from Čeština 2.0 are in `src/dev_testing/build_data`. Scripts for conversion of the data to several different data representations for neural networks are in `scripts/data_repre`. Scripts for conversion between our data format and SIGMORPHON data format are in `scripts/data_conversion`. Finally, the `Makefile` that provides a simple access to some of the data processing steps lies in the root directory.

Evaluation Scripts for computing the accuracies and evaluating the models are located in `src/dev_testing/eval_models`. Checking, whether a predicted form is considered correct or not, and the specific accuracies are defined in `accuracies.py`. The main evaluation script for evaluating the copy baseline, skloňuj.cz model and the retrograde model, is `main.py`. The predictions in the data format for neural networks (produced by seq2seq models) are evaluated using the script `evaluate_onmt_model.py`.

Models The retrograde model, and the baselines (skloňuj.cz, SIGMORPHON non-neural and copy baseline) are in `src/models`. The development configuration files of the OpenNMT models are in `configs`, and the scripts for running the jobs on cluster to train and evaluate the OpenNMT models lie in `scripts`. Configuration files of specific models can be found in the log directory.

Data We omit most of the data because it can be built automatically. The only data we include are the following: parts of files from the process of building the test-oov dataset – automatically inflected and manually corrected version, both in `data/processed` – and the training, development, test and test-oov datasets, all in `data/cleaned`. The training, development and test set are publicly available and licensed under Creative Commons - Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0). They were automatically extracted from [Hajič et al., 2020]. The test-oov dataset

was created manually from data from the project čeština2.0[3] and they are licensed under the same license.

Logs In the log directory `data/log/evaluate_models/onmt/experiments` there are logs from experiments run by the OpenNMT library. To save space, we omit logs from most of the experiments. However, logs from the final experiments with our 2 final models (one Transformer and one LSTM-based) are there as an example of log directories. The original config file that was used for the experiment is always saved in `config.yaml`. The `*predictions.txt` files (if present) contain raw predictions of the models for different datasets. The `*.pred` (if present) files contain nicely formatted predictions, and the `*.res` files (if present) contain the evaluated predictions together with corrected errors, as described in Section 5.1.2. The files `training_accs.*` contain logs about accuracies during training: both on the training and the development data (evaluated only on a subset of the large datasets containing 1k lemmata).

Predictions on test-oov We especially mention the location of prediction files on the test-oov datasets. The files are in the log directory `data/log/evaluate_models/onmt/experiments`, in the subfolders corresponding to the models included in the final evaluation (see Section 5.5).

Other The scripts for the inflection library are in `inflection_lib` (notes on how to install and use it are in Chapter 7.

In `data/inner/sklonuj_cz.php` lies the original script for skloňuj.cz. It was provided to us 30th of January, 2022 by the maintainers of the web, under the license GNU Lesser General Public License 2.1. The version on web[4] has probably been updated since that date and therefore can provide different inflection than the system we use.

An example of nouns from MorfFlex with unusual tag sets (as described in data filtering section 3.3.2) as in `data/log/strange_tagsets.log`.

6.2 Technical details

The project was developed on Linux Mint system. Most of the scripts are written in `python3` or in `bash` (simple processing scripts and running cluster jobs). The main `python` library used in the project is OpenNMT-py.

[3] `https://cestina20.cz/`

[4] `https://sklonuj.cz/`

To prepare the whole development directory on a Linux system, you should perform the following steps:

1. Create a `python3` virtual environment in the root directory of the development repository (`cz-inflect-dev/.venv`) , e.g. by running `mkdir -p .venv && python3 -m venv .venv`.
2. Run `make .venv` to install the project and the requirements.
3. Run `make build_data`, if you want to build the data from scratch (not needed to run the retrograde model).

The `Makefile`, present in the root directory, controls the installation of dependencies (to be able to do it automatically it is needed that the name of the virtual environment folder is specifically `.venv`) and data downloading and processing.

After running `make build_data` it first installs all the dependencies. Then it downloads MorfFlex, extracts nouns only, shuffles the data, performs filtering and other processing of the data, removes duplicates, builds the datasets and converts them to the format for neural network models. The individual scripts providing the processing lie in `src/morfflex`.

To be able to run the skloňuj.cz baseline, you need to have installed `PHP 7.3` and additionally package `php7.3-mbstring`.

6.2.1 Run the models

In this section we briefly describe how to run the retrograde model and the Transformer model TRM-11.

Retrograde model To run the retrograde model, you also need to have installed the requirements and the project. Then you can simply run the interactive script `retrograde_model.py`, which lies in `src/czech_inflection/ /models/retrograde_model/`. It will take some time to load the model, but then it performs inflection relatively quickly. The script itself shows how to use the retrograde model as a library.

TRM-11 The Transformer model lies in the development directory of the inflection library: `inflection_lib/cz-inflect/`. To run the library script with the TRM-11 instead of LSTM-44, you need to change the path to the model in the script `inflect.py` (self-descriptive). If you have already installed the requirements and the project by running `make .venv`, then you

can run it from the root development directory (`cz-inflect-dev`) by running `.venv/bin/python3 inflection_lib/cz-inflect/inflect.py`.

Run on GPU To run the inflection model from the inflection library on GPU instead of CPU, you need to modify the script `inflect.py`, specifically the definition of variable `opt` in method `_inflect_file()` (parameter `gpu`).

6.3 Data format

In our work, we use 2 different data formats: the standard one and the one for neural networks models.

6.3.1 Standard

In the standard format, there is one entry (row) for each paradigm table, and it is semicolon-separated list of lemma and its 14 forms (the row has 15 items: the lemma, the singular forms and the plural forms). The following is a shortened example for lemma *dítě* (child):

```
dítě;dítě;dítěte;dítěti;dítě;dítě;dítěti;dítětem;...;dětmi
```

Non-existent forms are marked by a special token "`?`" (an example for lemma *nůžky* (scissors)):

```
nůžky;?;?;?;?;?;?;?;nůžky;nůžek;nůžkám;...;nůžkami
```

In the test-oov dataset, multiple forms in one paradigm cells are allowed. The character "/" serves to separate multiple forms (an example for lemma *muž* (man), which is not in the test-oov dataset):

```
muž;muž;muže;muži/mužovi;muže;muži;muži/mužové;mužích;...
```

6.3.2 Data for seq2seq models

In the data format for the seq2seq models there is a separate entry for every form. Morphological information about the form is encoded to a simple 2-positional tag (first position is number, S=singular, P=plural), second position is the case (1 up to 7). The source sequence is sharp-separated lemma and tag, the target sequence is the form itself. Both source and target

lemma with tag	form
m a t k a # S 1	m a t k a
m a t k a # S 2	m a t k y
m a t k a # S 3	m a t c e
m a t k a # S 4	m a t k u
m a t k a # S 5	m a t k o
m a t k a # S 6	m a t c e
m a t k a # S 7	m a t k o u
m a t k a # P 1	m a t k y
m a t k a # P 2	m a t e k
m a t k a # P 3	m a t k á m
m a t k a # P 4	m a t k y
m a t k a # P 5	m a t k y
m a t k a # P 6	m a t k á c h
m a t k a # P 7	m a t k a m i

Table 6.1 Data format for seq2seq models: example of all 14 entries corresponding to lemma *matka* (mother).

sequences are tokenized into individual characters. An example is in Table 6.1.

6.4 Evaluation

`src/dev_testing/eval_models/` The computing of accuracies is implemented in `accuracies.py`.

6.5 Retrograde model

In the following paragraphs, we describe more profoundly how the Retrograde model works and how it is implemented. (For information of how to run it, refer to 6.2.1.)

6.5.1 Retrograde trie

To search effectively in the whole morphological dictionary, we use a datastructure that we call *retrograde trie*. It is based on a *multi-value trie*, which is a standard character-based trie with the modification that we can store multiple values for one key. It supports the following operations:

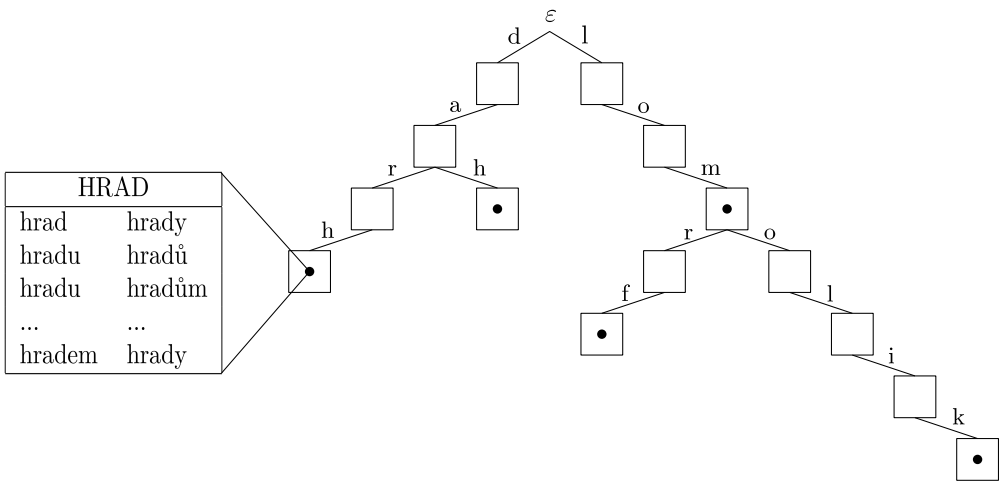

Figure 6.1 Retrograde trie: example. The trie currently contains five lemmata (hrad, hrad, frmol, mol and kilomol). The dot inside the node represents the lemma, which is saved there as a key together with its paradigm table (showed only for lemma hrad).

- add a new *value* to a node corresponding to a *key*
- for a given node, return all values that are stored in the corresponding sub-tree (the subtree is traversed by depth first search, the value from a node is returned before traversing its subtree)
- for a new key, search for the longest common prefix contained in the trie and return all values stored in the prefix sub-tree

Retrograde trie works the same, only it stores the keys as reversed strings. Therefore it allows finding the longest common suffix (ending segment) instead of the prefix.

Lemmata from the morphological dictionary are used as keys, and whole corresponding paradigms are used as values. For some lemmata, there can be several different paradigms (homonyma that are equal in the base form but differ in the inflected forms). That is the reason for using multi-value trie.

We build the retrograde trie at the time of model initialization by simply adding the lemma-paradigm tuples one by one. During prediction we retrieve paradigms of the lemmata that have longest common suffix and use them as paradigms.

6.5.2 Method for inflection according to a paradigm

To inflect lemma X according to the paradigm of lemma A (with a known paradigm), we perform the following. First, split both lemmata into two parts:

stem and suffix. These stems and suffixes are technical only and generally can be distinct from the morphological stem and suffix of the lemmata. We perform the split such that the suffix is the same for both lemmata (and the longest possible one), and stem is the rest of the word. Then we take the whole paradigm for lemma A and in each form we replace the A-stem with the X-stem, obtaining inflected forms of the lemma X.

Sometimes it can happen that the inflected form of lemma A does not contain the whole A-stem. If the suffix is not long enough, there could be some changes inside the stem when creating the inflected form from the lemma.

To make the approach work also in this case, in each form of the A-paradigm we remove the prefix that has the same length as the A-stem, replacing it with the X-stem. If the A-form is the same length as A-stem or shorter (it is not possible to remove prefix of that length), we just replace the whole A-form with the X-stem.

If there were any non-existent forms in the original paradigm table, we preserve them as non-existent in the predicted paradigm table.

If the length of the longest common suffix is zero, we do not perform any inflection according to a paradigm. In such case we consider the lemma as inflexible and produce corresponding number of forms that are equal to the lemma itself.

6.5.3 Prediction combination

To retrieve the paradigms from the retrograde trie, we traverse the subtree of the node corresponding to the longest common suffix, yielding all paradigms present in the subtree. If the maximal number of lemmata allowed to combine is given, we traverse the subtree until the maximal count is reached. Since the traversing is done by DFS, lemmata with the same character at specific position are returned consecutively. The children of a node corresponding to individual characters are visited in the order they appeared in the datastructure, which is random (because the trie was built by adding randomly shuffled data one by one).

Once we have a set of predicted paradigm tables according to all the paradigms, we combine the predictions to obtain a single paradigm table.

For each paradigm cell, we take the most common form from the suggested corresponding cells in all the paradigms. In case of a tie, the answer is chosen randomly among the most frequent predictions. If it is the non-existent form, we take the second most common form. If there are only non-existent forms amongst the suggested forms, we replace the form with the lemma itself. By that we prevent the model from predicting the non-existent form, following the trend that some suggestion is better than no suggestion.

6.5.4 Possible improvements

There are several things that could be improved in the retrograde model.

- Inflection according to a paradigm: when the stem is not the same for all the forms of the paradigm A, try to modify the X-stem in a way in which the A-stem is modified.
- If there is an non-existent form in the final predicted paradigm table, try to take shorter longest common suffix and inflect according to words sharing the new common suffix. - This would make it work relatively reliable also in the setting where we would have a lot of gaps in the training paradigm tables.
- Pre-compute the suffix tables for every node of the trie and during prediction just find the node with longest common suffix and inflect the lemma according to the suffix table saved in the node. The motivation is clear: if we build the model and then repetitively use it to inflect the same word, it is not efficient: once we are in a specific node of a trie corresponding to the longest common suffix, the inflection table we get is independent of the word we are trying to inflect. This would save space for saving the model and time for prediction.

6.6 Building the test-oov dataset

`src/dev_testing/build_data`

In `cestina20reader.py` we extract the relevant parts of the database (lemma and explanations) of neologism provided to us, and in `build_data.py` we process the data more.

The first part of the script (called by `python3 build_data.py --inflect`) removes word phrases and selects OOV words only.

We use processed MorfFlex (extracted raw lemmata only) to determine whether a given noun is OOV or not (`src/morfflex/lexicon.py`).

The script `build_data.py -inflect` then shuffles the lemmata randomly and then uses the rule-based baseline model (skloňuj.cz) to automatically inflect the lemmata. The inflected lemmata with explanations are pretty-printed to a file to be checked manually (`data/processed/6-inflected.txt`).

Then we are expected to perform the manual check: fix the incorrectly inflected forms, add multiple options of correctly inflected forms if there are more than a single one, mark the faulty lemmata and mark the end of the checked part of the file.

After the manual check, the second part of the script (`build_data.py --load_checked`) loads the data from the manually edited text file, removes

the lemmata marked as faulty and removes the explanations to obtain the final test-oov data (`data/cleaned/test_oov_data.txt`).

Chapter 7

User guide

This is a user guide that briefly explains how to use the inflection library provided in the attachment, in directory `cz-inflect`[1].

The library provides a morphological guesser for inflection of Czech nouns. It focuses on inflection of out-of-vocabulary words. (For other words we recommend MorphoDiTa[2] tool.)

The guesser is a LSTM-based encoder-decoder architecture with attention, trained with OpenNMT-py library on a training dataset consisting of approx. 360k Czech noun lemmata (data source: MorfFlex2.0 [Hajič et al., 2020]).

7.1 Build

To run the inflection script, you need Python3[3] and a python environment with OpenNMT[4] library installed.

Linux On Linux systems (tested on Mint), you can create the python virtual environment and install the dependencies using the provided script `build.sh`:

```
unzip cz-inflect.zip
cd cz-inflect
bash build.sh
```

Otherwise Otherwise you have to install the python dependencies listed in the file `requirements.txt` manually.

[1] `https://github.com/tomsouri/cz-inflect/releases/tag/BP_official`
[2] `https://lindat.mff.cuni.cz/services/morphodita/`
[3] tested with Python 3.8.10
[4] OpenNMT-py==3.0.4

7.2 Run

After building the environment by the previous step, you can use the provided script `run.sh` or simply run `.venv/bin/python3 inflect.py` (if you installed the dependencies manually, provide proper path to the python binary file). This runs the inflection script.

The script reads lemmata from the standard input (one lemma per line) and prints the tab-separated inflected forms to standard output (14 forms per line).

7.3 Examples

The following is an example of an interactive call with lemma *dub* (oak):

```
> bash run.sh
Enter a lemma to be inflected:
dub
dub dubu dubu dub dube dubu dubem duby dubů dubům duby duby
dubech duby
...
```

An example of a non-interactive call (lemma *buk* (beech)):

```
> echo "buk" | bash run.sh
buk buku buku buk buku buku bukem buky buků bukům buky buky
bucích buky
```

7.4 Python library

Additionally, the program can be used as a Python library. It provides the method `inflect` that receives a single lemma (or a list of lemmata) and returns corresponding inflected forms. The simplest use is demonstrated in listing 1 (the content of the script `example_usage.py`).

7.5 Notes on performance

The inflection model focuses on inflection of out-of-vocabulary words and it is a guesser. It is not designed to substitute dictionary-based systems but rather to complement them. The inflection it provides is not perfect, especially for

Listing 1 Example program using the inflection library.
Inflection of words *lingebra* (short for linear algebra), *programko* (short for programming) and *matfyz* (short for the Faculty of Mathematics and Physics)

```
#!/usr/bin/env python3
from inflect import inflect

# Inflecting a single lemma

lemma = "lingebra"
inflected_forms = inflect(lemma)

print(f"Inflected forms of lemma {lemma}:")
print(", ".join(inflected_forms))


# Inflecting a list of lemmata

lemmata = ["programko", "matfyz"]
infl_lemmata = inflect(lemmata)

for (lemma, inflected_forms) in zip(lemmata, infl_lemmata):
  print(f"Inflected forms of lemma {lemma}:")
  print(", ".join(inflected_forms))
```

words that inflect unusually (e.g. for lemma *pes* (dog) the script outputs *pes, pesu, psi, pes, pese, psi, ...*

The model was trained on data containing entries with non-existent forms as targets (marked as "?" in the data). Therefore it can sometimes produce the token "?" instead of a predicted form, suggesting that the specific form probably does not exist.

Due to long time spent by loading the model, the inflection of the first word takes relatively long time (~10s).

The inflection model is case-sensitive.

The model is not able to correctly inflect neither phrases (automatically deletes spaces from the input), nor words containing special characters (e.g. "-", "=" etc.) – it substitutes them by some alphanumeric characters.

The model prints some messages to standard error during prediction. It looks like `[<DATE> INFO] PRED SCORE: -0.0014, ... NB SENTENCES: 14` and it is not part of the predictions.

Conclusion

This thesis focused on the task of automatic morphological inflection in Czech, specifically on inflection of nouns in out-of-vocabulary (OOV) conditions.

We achieved the following:

- We automatically extracted a large train-dev-test dataset of Czech inflected nouns from the MorfFlex dataset [Hajič et al., 2020], aimed at evaluation in the OOV conditions.
- We manually built small real-world OOV test dataset of neologisms (test-oov).
- We developed three different inflection systems: a retrograde model, simple to implement and understand, and two seq2seq models, one LSTM- and one Transformer-based.
- We compared our systems to one publicly available ready-to-use rule-based inflection system and three standard baselines from SIGMORPHON shared tasks
- Our seq2seq models achieved particularly good results on the standard test dataset, the Transformer beating all other models.
- The retrorade model showed significant ability to inflect real-world OOV words, outperforming all neural models on the test-oov dataset.
- We trained and evaluated one of our best setups on SIGMORPHON 2022 shared task data (18 development languages, Czech was not included, large training data condition), and achieved state-of-the-art results in the OOV evaluation condition (feature overlap) on 6 languages.
- We achieved state-of-the-art results on the same dataset in the both overlap condition (test pair's feature set and lemma are both attested in training, but not together in one entry) on 8 languages.

- We released a Python library[5] with ready-to-use inflection model for Czech nouns, which could (especially once extended to another parts-of-speech) work as as complementary system to MorphoDiTa [Straková et al., 2014] - as a back-off for OOV words. The released system performs significantly better than the tested system skloňuj.cz, which (together with other similar rule-based systems) represents the only option for OOV inflection in Czech publicly available on web.

There is still room for improvement. The hyperparameters of the best model we tested (TRM-11) was tuned only a little and we suppose to be able to achieve better results when tuning more.

All neural models performed really poorly on the real-world test-oov dataset, compared to the non-neural models. Nevertheless, also the non-neural systems showed relatively weak performance on that dataset. This indicates that the real-world OOV words may morphologically differ from the words randomly sampled from the MorfFlex dictionary, and thus the used training dataset (unweighted dictionary words) may be suboptimal for learning paradigms of neologisms. It would be beneficial to investigate this issue more.

The released library could be improved too. The system was trained on data containing entries for non-existent forms and therefore it can sometimes generate the special token representing such forms. Moreover, it is not able to deal with non-alphanumeric characters correctly (which could appear in OOV words, e.g. e-book etc.).

We summarize the possible future development and research direction in the following section.

Future work

We divide the possible future direction based on the topic they are connected to, into three sections: data, task broadening and further investigation of current approach.

Data

- Investigate more the nature of OOV words: what are the common OOV words? Build a corresponding dataset.

- Neologisms: investigate more whether it is sensible to use all non-OOV words to train a system for inflection of OOV words. Do neologisms

[5] `https://github.com/tomsouri/cz-inflect/releases/tag/BP_official`

behave morphologically the same (or similarly) as all the other words in Czech? Are they rather morphologically similar to newer words (neologisms that are already in the morphological dictionaries)? Or to the less frequent words? Or to some other class of words?

- Investigate more the limitations of the approach to data filtering.
- Is a plentiful dataset needed for training our models? Are all training entries needed? Some research could be conducted to explore how the performance drops when randomly sampling only a smaller part of the dataset for training. Smaller (yet representative enough) dataset would lead to the possibility of training for more epochs easily.

Broadening of the task

- Inflect not only nouns, but all flexible parts-of-speech: define new principles of how should the inflection task work, make design decisions for dataset building, build appropriate datasets. Develop and evaluate new models, and finally release a ready-to-use system capable of inflecting all parts-of-speech: the true complementary system (back-off for OOV words) to MorphoDiTa [Straková et al., 2014].
- Train the models on data from other languages, with the goal of not only evaluating the performance, but also building a reliable model. E.g. for Slovak, it should be possible to build datasets similarly to our approach (there is a Slovak version of MorfFlex morphological dictionary [Hajič and Hric, 2017]). Release a ready-to-use model for inflection in Slovak.
- Deal with ambiguous lemmata (homonyms): find a way to be able to deal correctly with homonyms.
- Build and train a model for inflection of words with removed diacritics.

Further investigation of current approaches

- Perform further hyperparameter tuning with our seq2seq models, especially with the Transformer architecture (model TRM-11, which was tuned very little).
- Try model ensembling with our seq2seq models, to see whether it leads to further improvement.

- Add more morphological information to the tag (in data for seq2seq models), such as gender. During prediction of whole paradigm table for a given lemma, first use a morphological analyser to get the possible morphological categories of the lemma, and for the most probable ones perform the actual prediction of inflected forms (adding the additional morphological information to the input of the inflection generator). This could theoretically lead to a solution of dealing homonyms, at least in the case that they differ in some morphological category.

- Investigate more the fact that OpenNMT uses a per-token loss and we aim at per-sequence accuracy. Is using the OpenNMT library the best approach?

- Improve the released model to work better in real-world use cases (ability of correctly dealing non-alphanumeric characters in the input lemma, not predicting the non-existent-form token for any lemma).

- Optimize the retrograde model with regard to disk space usage and prediction speed: pre-extract the abstract paradigm tables, do not hold the full paradigm tables for the lemmata in the nodes of the trie, and during prediction simply inflect according to the one abstract paradigm table.

- Investigate more the poor results on the real-world OOV dataset (test-oov). Inspect the actual predictions of the individual systems and try to find an explanation of the behaviour. Check to what extent the systems make similar errors. Look for consistent error patterns. Analyze how the test-oov words differ from words in MorfFlex. Check whether and how frequently the inflectional paradigms of the test-oov words are present in MorfFlex. Check to what extent the frequency of inflectional paradigms differs between MorfFlex and test-oov.

Final remark We hope to be able to follow some of the directions in our future research, especially building a model for inflection of all parts of speech, and developing a model for Slovak.

Bibliography

Aharoni, Roee et al. [Aug. 2016]. "Improving Sequence to Sequence Learning for Morphological Inflection Generation: The BIU-MIT Systems for the SIGMORPHON 2016 Shared Task for Morphological Reinflection". In: *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Berlin, Germany: Association for Computational Linguistics, pp. 41–48. DOI: 10.18653/v1/W16-2007. URL: https://aclanthology.org/W16-2007.

Bahdanau, Dzmitry et al. [2016]. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv: 1409.0473 [cs.CL].

Bergmanis, Toms et al. [Aug. 2017]. "Training Data Augmentation for Low-Resource Morphological Inflection". In: *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*. Vancouver: Association for Computational Linguistics, pp. 31–39. DOI: 10.18653/v1/K17-2002. URL: https://aclanthology.org/K17-2002.

Chung, Junyoung et al. [2014]. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. arXiv: 1412.3555 [cs.NE].

Cotterell, Ryan et al. [Aug. 2016]. "The SIGMORPHON 2016 Shared Task—Morphological Reinflection". In: *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Berlin, Germany: Association for Computational Linguistics, pp. 10–22. DOI: 10.18653/v1/W16-2002. URL: https://aclanthology.org/W16-2002.

Cotterell, Ryan et al. [Aug. 2017]. "CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection in 52 Languages". In: *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*. Vancouver: Association for Computational Linguistics, pp. 1–30. DOI: 10.18653/v1/K17-2001. URL: https://aclanthology.org/K17-2001.

Cotterell, Ryan et al. [Oct. 2018]. "The CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection". In: *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*.

Brussels: Association for Computational Linguistics, pp. 1–27. DOI: 10.18653/v1/K18-3001. URL: https://aclanthology.org/K18-3001.

Durrett, Greg and John DeNero [June 2013]. "Supervised Learning of Complete Morphological Paradigms". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, pp. 1185–1195. URL: https://aclanthology.org/N13-1138.

Dušek, Ondřej and Filip Jurčíček [Aug. 2013]. "Robust multilingual statistical morphological generation models". In: *51st Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 158–164. URL: https://aclanthology.org/P13-3023.

Faruqui, Manaal et al. [2016]. *Morphological Inflection Generation Using Character Sequence to Sequence Learning*. arXiv: 1512.06110 [cs.CL].

Goldman, Omer et al. [May 2022]. "(Un)solving Morphological Inflection: Lemma Overlap Artificially Inflates Models' Performance". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Dublin, Ireland: Association for Computational Linguistics, pp. 864–870. DOI: 10.18653/v1/2022.acl-short.96. URL: https://aclanthology.org/2022.acl-short.96.

Hajič, Jan and Jan Hric [2017]. *MorfFlex SK 170914*. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-3277.

Hajič, Jan et al. [June 2009]. "The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages". In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*. Boulder, Colorado: Association for Computational Linguistics, pp. 1–18. URL: https://aclanthology.org/W09-1201.

Hajič, Jan et al. [2020]. *Prague Dependency Treebank - Consolidated 1.0 (PDT-C 1.0)*. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-3185.

Hajič, Jan et al. [2020]. *MorfFlex CZ 2.0*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-3186.

Hochreiter, Sepp and Jürgen Schmidhuber [1997]. "Long Short-Term Memory". In: *Neural Comput.* 9.8, 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.

1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Kann, Katharina and Hinrich Schütze [Aug. 2016]. "MED: The LMU System for the SIGMORPHON 2016 Shared Task on Morphological Reinflection". In: *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Berlin, Germany: Association for Computational Linguistics, pp. 62–70. DOI: 10.18653/v1/W16-2010. URL: https://aclanthology.org/W16-2010.

— [Aug. 2017]. "The LMU System for the CoNLL-SIGMORPHON 2017 Shared Task on Universal Morphological Reinflection". In: *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*. Vancouver: Association for Computational Linguistics, pp. 40–48. DOI: 10.18653/v1/K17-2003. URL: https://aclanthology.org/K17-2003.

Kann, Katharina and Hinrich Schütze [2016]. *Single-Model Encoder-Decoder with Explicit Morphological Representation for Reinflection*. arXiv: 1606.00589 [cs.CL].

Kingma, Diederik P. and Jimmy Ba [2017]. *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Klein, Guillaume et al. [July 2017]. "OpenNMT: Open-Source Toolkit for Neural Machine Translation". In: *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, pp. 67–72. URL: https://www.aclweb.org/anthology/P17-4012.

Kodner, Jordan et al. [July 2022]. "SIGMORPHON–UniMorph 2022 Shared Task 0: Generalization and Typologically Diverse Morphological Inflection". In: *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Seattle, Washington: Association for Computational Linguistics, pp. 176–203. DOI: 10.18653/v1/2022.sigmorphon-1.19. URL: https://aclanthology.org/2022.sigmorphon-1.19.

Králíková, Květoslava and Jarmila Panevová [1990]. "ASIMUT - A Method for Automatic Information Retrieval from Full Texts". In: *Explizite Beschreibung der Sprache und automatische Textbearbeitung* XVII.

Levenshtein, Vladimir I. [1965]. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics. Doklady* 10, pp. 707–710.

Liu, Ling and Mans Hulden [July 2020]. "Leveraging Principal Parts for Morphological Inflection". In: *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Online: Association for Computational Linguistics, pp. 153–161. DOI: 10.18653/v1/2020.sigmorphon-1.17. URL: https://aclanthology.org/2020.sigmorphon-1.17.

Liu, Ling and Mans Hulden [2021]. "Can a Transformer Pass the Wug Test? Tuning Copying Bias in Neural Morphological Inflection Models". In: *CoRR* abs/2104.06483. arXiv: 2104.06483. URL: https://arxiv.org/abs/2104.06483.

Luong, Minh-Thang et al. [2015]. *Effective Approaches to Attention-based Neural Machine Translation*. arXiv: 1508.04025 [cs.CL].

Makarov, Peter and Simon Clematide [Oct. 2018]. "UZH at CoNLL–SIGMORPHON 2018 Shared Task on Universal Morphological Reinflection". In: *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*. Brussels: Association for Computational Linguistics, pp. 69–75. DOI: 10.18653/v1/K18-3008. URL: https://aclanthology.org/K18-3008.

McCarthy, Arya D. et al. [Aug. 2019]. "The SIGMORPHON 2019 Shared Task: Morphological Analysis in Context and Cross-Lingual Transfer for Inflection". In: *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Florence, Italy: Association for Computational Linguistics, pp. 229–244. DOI: 10.18653/v1/W19-4226. URL: https://aclanthology.org/W19-4226.

Merzhevich, Tatiana et al. [July 2022]. "SIGMORPHON 2022 Task 0 Submission Description: Modelling Morphological Inflection with Data-Driven and Rule-Based Approaches". In: *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Seattle, Washington: Association for Computational Linguistics, pp. 204–211. DOI: 10.18653/v1/2022.sigmorphon-1.20. URL: https://aclanthology.org/2022.sigmorphon-1.20.

Mikulová, Marie et al. [2020]. *Manual for Morphological Annotation. Revision for Prague Dependency Treebank – Consolidated 2020 release*. Vol. ÚFAL TR-2020-64. The ÚFAL/CKL Technical Report Series (ISSN 1214-5521). Institute of Formal, Applied Linguistics (ÚFAL), Faculty of Mathematics, and Physics, Charles University. URL: https://ufal.mff.cuni.cz/techrep/tr64.pdf.

Pimentel, Tiago et al. [Aug. 2021]. "SIGMORPHON 2021 Shared Task on Morphological Reinflection: Generalization Across Languages". In: *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Online: Association for Computational Linguistics, pp. 229–259. DOI: 10.18653/v1/2021.sigmorphon-1.25. URL: https://aclanthology.org/2021.sigmorphon-1.25.

Ruder, Sebastian [2016]. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747*.

Slavíčková, Eleonora [1975]. *Retrográdní morfematický slovní češtiny*. 1. Academia.

Sorokin, Alexey [Aug. 2016]. "Using longest common subsequence and character models to predict word forms". In: *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Berlin, Germany: Association for Computational Linguistics, pp. 54–61. DOI: 10.18653/v1/W16-2009. URL: https://aclanthology.org/W16-2009.

Straková, Jana et al. [June 2014]. "Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, pp. 13–18. DOI: 10.3115/v1/P14-5003. URL: https://aclanthology.org/P14-5003.

Sutskever, Ilya et al. [2014]. *Sequence to Sequence Learning with Neural Networks*. arXiv: 1409.3215 [cs.CL].

Vaswani, Ashish et al. [2017]. *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Vylomova, Ekaterina et al. [July 2020]. "SIGMORPHON 2020 Shared Task 0: Typologically Diverse Morphological Inflection". In: *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Online: Association for Computational Linguistics, pp. 1–39. DOI: 10.18653/v1/2020.sigmorphon-1.1. URL: https://aclanthology.org/2020.sigmorphon-1.1.

Wu, Shijie et al. [2020]. "Applying the Transformer to Character-level Transduction". In: *CoRR* abs/2005.10213. arXiv: 2005.10213. URL: https://arxiv.org/abs/2005.10213.

— [Apr. 2021]. "Applying the Transformer to Character-level Transduction". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, pp. 1901–1907. DOI: 10.18653/v1/2021.eacl-main.163. URL: https://aclanthology.org/2021.eacl-main.163.

Yang, Changbing et al. [July 2022]. "Generalizing Morphological Inflection Systems to Unseen Lemmas". In: *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Seattle, Washington: Association for Computational Linguistics, pp. 226–235. DOI: 10.18653/v1/2022.sigmorphon-1.23. URL: https://aclanthology.org/2022.sigmorphon-1.23.

Yang, Shudong et al. [2020]. "LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example". In: *2020 International Workshop on Electronic Communication and Artificial*

Intelligence IWECAI, pp. 98–101. DOI: 10.1109/IWECAI50956.2020.00027.

Zhou, Chunting and Graham Neubig [Aug. 2017]. "Morphological Inflection Generation with Multi-space Variational Encoder-Decoders". In: *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*. Vancouver: Association for Computational Linguistics, pp. 58–65. DOI: 10.18653/v1/K17-2005. URL: https://aclanthology.org/K17-2005.

Šmerk, Pavel and Pavel Rychlý [2009]. *Majka – rychlý morfologický analyzátor*. cze. Tech. rep. Masarykova univerzita. URL: http://nlp.fi.muni.cz/ma/.

List of Figures

List of Tables

Appendix A

Full tables

Here we present full result of development experiments with the Retrograde model, and evaluation on SIGMORPHON 2022 shared task dataset in all evaluation conditions.

Model	Trainsize	Combine	FA	FPA
Copy-baseline			22.53	1.53
Retrograde	1	1	22.63	1.61
Retrograde	2	2	29.74	9.83
Retrograde	4	4	30.26	10.37
Retrograde	5	4	34.08	23.58
Retrograde	10	8	42.33	28.79
Retrograde	50	32	78.25	67.29
Retrograde	100	64	83.17	72.95
Retrograde	200	128	86.54	76.26
Retrograde	400	256	89.76	78.89
Retrograde	500	256	90.41	79.33
Retrograde	800	512	90.96	81.35
Retrograde	1k	512	91.12	81.66
Retrograde	2k	1024	91.92	83.16
Retrograde	5k	4096	92.31	83.95
Retrograde	10k	8192	92.91	84.83
Retrograde	20k	16384	93.21	85.40
Retrograde	40k	32768	93.65	86.28
Retrograde	80k	65536	94.01	87.00
Retrograde	120k	65536	94.26	87.54
Retrograde	150k	131072	94.37	87.70
Retrograde	175k	131072	94.40	87.78
Retrograde	200k	131072	94.45	87.90
Retrograde	225k	131072	94.49	87.95
Retrograde	250k	131072	94.52	88.08
Retrograde	280k	131072	94.55	88.14
Retrograde	320k	131072	94.63	88.32
Retrograde	360k	131072	**94.67**	**88.43**

Table A.1 Retro eval: combine=max number of paradigms for combination, FA=form accuracy, FPA=full paradigm accuracy, whole evaluation is on development data. We compare the performance with the copy-baseline.

		Submitted systems					Baselines		
Lang	Partition	CLUZH	Flexica	OSU	TüM Main	UBC	Neural	NonNeur	OUR
ang	overall	**64.9**	41.1	44.5	60.9	60.0	61.1	43.1	60.3
	both	82.5	73.2	80.5	82.1	80.9	**83.1**	78.5	81.9
	lemma	**48.4**	11.7	10.8	42.5	41.8	41.0	10.8	39.6
	features	**76.6**	64.4	73.7	71.9	74.1	73.4	68.7	**76.6**
	neither	**53.2**	14.5	12.7	45.7	39.3	48.0	12.7	45.1
ara	overall	75.9	37.5	40.9	75.3	67.2	**78.5**	26.9	74.3
	both	80.0	66.3	80.9	81.6	74.3	81.2	52.8	**81.8**
	lemma	73.9	10.4	1.3	71.8	71.1	**77.3**	1.3	68.4
	features	81.7	65.5	78.7	78.5	65.5	**81.9**	50.8	79.9
	neither	67.9	7.9	2.8	68.9	56.2	**73.6**	2.8	66.8
asm	overall	70.7	34.3	43.5	63.1	75.6	**76.8**	31.9	66.8
	both	**90.8**	68.7	86.3	77.2	85.4	83.9	62.6	86.5
	lemma	50.9	0	1.1	49.1	65.8	**69.7**	1.1	46.9
	features	83.3	75.0	75.0	**91.7**	83.3	83.3	83.3	83.3
	neither	33.3	0	0	22.2	**88.9**	77.8	0	**88.9**
evn	overall	48.9	3.8	25.0	52.0	57.5	**57.7**	25.1	47.7
	both	**66.7**	**66.7**	0	**66.7**	**66.7**	**66.7**	**66.7**	**66.7**
	lemma	40.4	1.9	12.6	45.6	52.4	**53.4**	12.6	41.4
	features	-	-	-	-	-	-	-	-
	neither	62.4	6.7	44.6	62.1	**65.5**	64.4	44.6	57.5
got	overall	65.7	21.3	51.3	65.3	**73.4**	72.2	46.0	66.8
	both	95.5	38.2	**95.9**	93.3	95.8	95.8	84.6	94.9
	lemma	35.7	3.5	4.7	38.2	**52.2**	49.6	4.7	39.6
	features	92.9	41.4	**94.1**	91.7	91.7	93.5	87.6	91.7
	neither	40.0	5.4	17.1	36.1	**50.2**	47.3	17.1	38.5
heb	overall	**51.8**	28.0	50.0	47.9	44.0	48.5	20.4	47.9
	both	94.1	55.9	94.4	94.4	86.5	**96.6**	35.1	95.1
	lemma	**9.4**	0.1	5.6	1.4	1.4	0.3	5.6	0.7
	features	-	-	-	-	-	-	-	-
	neither	-	-	-	-	-	-	-	-
hun	overall	72.3	33.0	47.1	68.2	74.9	**77.2**	37.2	67.5
	both	94.8	64.3	94.2	94.5	93.8	94.8	75.0	**95.1**
	lemma	54.6	2.5	1.3	45.4	60.0	**61.9**	1.3	46.0
	features	93.5	62.9	93.1	92.8	91.5	94.4	73.1	**94.5**
	neither	49.1	2.6	0.6	41.9	56.5	**59.0**	0.6	37.5
kat	overall	74.3	45.1	52.4	78.8	83.2	**87.2**	45.5	80.2
	both	95.1	79.3	94.6	96.0	**98.3**	97.4	77.7	96.0
	lemma	53.0	7.6	9.3	61.8	69.0	**77.2**	9.3	63.7
	features	96.7	95.7	96.7	96.7	96.7	**97.3**	96.7	96.7
	neither	54.8	9.5	12.5	60.7	65.5	**76.8**	12.5	66.7
khk	overall	47.9	23.4	**49.2**	47.7	46.3	49.1	38.0	48.9
	both	95.5	46.6	97.7	95.2	92.3	**98.1**	75.1	97.6
	lemma	0	0	**0.5**	0	0	0	**0.5**	0
	features	**94.1**	47.1	**94.1**	**94.1**	88.2	**94.1**	88.2	**94.1**
	neither	0	0	0	0	0	0	0	0

Table A.2 SIGMORPHON 2022 comparison – full results A: Partitioned results on large training (`ang-khk`). No feature overlap `evn` items and no feature overlap or both overlap `heb` items were included in the test set. Except for results of our system (OUR=LSTM-40), the table was taken from Kodner et al. [2022, Tables 17, 18].

		Submitted systems					Baselines		
Lang	Partition	CLUZH	Flexica	OSU	TüM Main	UBC	Neural	NonNeur	OUR
kor	overall	51.8	33.2	30.0	47.6	54.7	**56.2**	32.3	47.4
	both	**79.0**	67.5	61.7	69.3	76.2	78.7	66.1	72.8
	lemma	25.9	0.9	0	28.0	35.4	**36.9**	0	26.4
	features	**71.1**	55.4	50.6	56.6	60.2	62.7	59.0	39.8
	neither	27.1	0	0	20.0	**31.4**	18.6	0	12.9
krl	overall	58.4	37.9	45.2	24.1	**64.4**	27.1	5.4	61.0
	both	88.6	72.3	87.8	30.0	88.1	31.5	4.5	**89.3**
	lemma	27.3	2.1	0.9	8.6	**39.8**	13.7	0.9	31.1
	features	**87.5**	69.8	85.9	57.8	85.4	57.8	20.8	**87.5**
	neither	33.7	13.0	13.0	32.1	**48.4**	35.3	13.0	42.4
lud	overall	73.1	89.2	**89.7**	50.5	72.4	53.0	89.4	83.8
	both	94.8	95.9	**96.8**	96.0	94.7	96.5	95.9	94.5
	lemma	21.2	**51.5**	**51.5**	11.1	39.1	20.2	**51.5**	49.5
	features	87.3	92.0	92.9	93.4	88.2	**94.3**	93.4	83.5
	neither	66.6	**97.1**	**97.1**	3.3	56.9	5.6	**97.1**	86.7
non	overall	76.9	47.2	48.0	79.8	**87.2**	85.0	37.3	77.1
	both	90.8	68.7	90.5	89.8	**93.3**	92.4	68.0	91.4
	lemma	63.9	25.2	5.7	70.9	**82.1**	78.8	5.7	64.3
	features	85.2	77.0	85.2	80.3	**90.2**	88.5	80.3	83.6
	neither	51.4	25.7	17.1	57.1	**62.9**	51.4	17.1	37.1
pol	overall	86.5	52.9	47.8	67.7	**91.0**	69.5	43.6	87.2
	both	91.8	78.7	90.2	77.0	**95.1**	78.7	85.2	**95.1**
	lemma	84.3	15.7	0	71.4	87.1	68.6	0	**88.6**
	features	96.1	85.9	94.9	74.0	95.7	74.4	86.3	**96.7**
	neither	76.7	20.5	1.1	60.4	**86.1**	63.9	1.1	77.1
poma	overall	60.4	33.9	36.6	58.8	61.5	63.9	24.5	**64.7**
	both	73.4	48.5	74.6	69.2	69.8	**75.1**	40.8	73.4
	lemma	46.5	12.8	1.7	47.7	50.6	**59.9**	1.7	55.2
	features	**76.1**	54.5	70.1	69.4	73.3	74.1	47.8	74.7
	neither	44.9	14.6	2.4	48.4	50.2	52.2	2.4	**55.0**
slk	overall	85.5	58.2	47.4	65.8	**94.0**	70.1	47.5	86.0
	both	87.5	87.5	89.3	57.1	89.3	57.1	87.5	**92.9**
	lemma	89.4	44.7	2.1	51.1	**95.7**	57.4	2.1	87.2
	features	93.5	90.0	92.2	70.4	**95.7**	71.1	92.4	95.2
	neither	77.3	25.7	2.8	62.3	**92.4**	70.5	2.8	76.5
tur	overall	87.2	35.6	48.5	33.6	**94.2**	39.6	36.4	92.6
	both	97.9	72.7	96.2	36.0	98.4	37.4	72.7	**99.0**
	lemma	80.7	0.3	0.2	23.4	**93.3**	31.4	0.2	91.1
	features	93.7	57.9	**95.2**	80.2	92.9	79.4	66.7	**95.2**
	neither	52.7	0.8	5.3	40.5	**72.5**	71.0	5.3	57.3
vep	overall	57.5	30.5	36.9	44.1	**62.3**	48.8	32.4	58.6
	both	**75.5**	58.0	72.3	55.8	70.1	57.0	64.1	**75.5**
	lemma	42.8	1.4	1.4	25.9	**54.9**	33.9	1.4	44.4
	features	**71.5**	58.8	70.0	57.5	68.8	59.2	60.4	69.1
	neither	41.1	3.3	4.2	35.6	**55.4**	43.5	4.2	46.1

Table A.3 SIGMORPHON 2022 comparison – full results B: Partitioned results on large training (`kor-vep`). Except for results of our system (OUR=LSTM-40), the table was taken from Kodner et al. [2022, Table 18].