FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Kirill Semenov

Pre-processing of the Subword Encoding for the Neural Machine Translation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	Mgr. Martin Popel, Ph.D.
Study programme:	Computer Science
Study branch:	Language Technologies and Computational Linguistics

Prague 2024

Firstly, I would like to thank Martin Popel for encouraging me to take on the topic that combines the technical aspects of NLP and the linguistic analysis of tokenization. Since the field of NLP was relatively new to me, I am grateful for the attentive guidance in my work. This included both suggesting the directions of the research and constructive critique of my solutions, and finally his endless patience with the slow pace in my working progress.

However, all this research would have been virtually impossible without the help and support of the entire ÚFAL professorship. The year 2022 without exaggeration was the hardest in my life due to the political situation in Ukraine and Russia, which severely affected me. Contrary to that horrible background, the attitude towards me at ÚFAL during those days was supportive and I received nothing but understanding from the coordinators and professors of our program, including Markéta Lopatková, Vladislav Kuboň, Ondřej Bojar, Ondřej Dušek, Milan Straka, Daniel Zeman, Zuzana Biskupová, Eliška Zábrodská, and all other professors and teaching assistants at our institute. The fact that I am now writing it on the top of my Master's thesis is a direct result of their support, patience, and understanding of my situation, and I am sincerely grateful to them.

I would also like to thank the professors and the colleagues from the Charles University and from the research community for encouraging me to participate in the academic activities and for discussing their and my ideas of the research. In addition to the ÚFAL professors, I would like to especially point out Jiří Balhar, Vilém Zouhar, and Noëmi Aepli for fruitful communication. I am also thankful to my friends, both from my program and from outside of the university, as my time spent in Prague and at Charles was exciting and convenient. Anya, Katya, Zhenya, Jacobo, Daragh, Toni, Andy, Iqbal, Keenu, Natasha, Ulvi, Sonya, Doubravka, Roman - thank you for being there and sharing our time!

Finally, I would like to thank my family for supporting me throughout the course of my studies. Feeling acceptance from your closest ones is crucial in such long-term endeavors, and I am grateful to have that. Here, I would also like to thank my closest friends, especially Sasha, with whom we are now separated by the countries and the seas, but despite that we continue to lend each shoulder and share thoughts and feelings.

Нет войне!

Title: Pre-processing of the Subword Encoding for the Neural Machine Translation

Author: Kirill Semenov

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Martin Popel, Ph.D., Institute of Formal and Applied Linguistics

Abstract: In this thesis, we address the preprocessing approaches that improve the robustness of the subword tokenization for two types of noising.

We are focusing on the inline approaches to casing and diacritics in the texts, that is, allocating the casing and diacritics information to the special tokens that are separate from the words. In the field of casing noise, we compare the performance of our inline casing algorithm, InCa, and the existing solutions for inline case handling. We show that in some noising scenarios, our algorithm shows the best performance, and in the cases where it performs on par with the alternative solutions, the intrinsic parameters of the tokenizer trained on our data are more stable.

For the task of diacritics encoding, we are providing two solutions of inline diacritization, InDia, and show its improvement on robustness against the de-diacritized texts.

Since the final application that we plan to use our tokenizer is Czech-Ukrainian machine translation, we make a thorough comparison of the intrinsic and extrinsic performance of the inline approaches, and show that they have a correlation, although its scope is limited.

Keywords: Neural Machine Translation subword segmentation Unigram Language Model tokenization

Contents

Introduction **3**

1 Theoretical Background **6**
1.1 Tokenization: Main Approaches and Their Problems 6
1.1.1 Two Extremes: Words and Characters 6
1.1.2 Subword Segmentation: Fully Unsupervised and Language-Aware 7
1.1.3 Inherent Problems of Subword Tokenizers 8
1.2 Addressing the Typographic Variation by Tokenizers 9
1.2.1 Case Handling 9
1.2.2 Diacritics Handling 11
1.2.3 Dialectal Variation Within the Same Writing System . . . 12
1.2.4 Dialectal Variation with Different Writing Systems 12
1.3 Tokenization Metrics for NMT 13
1.3.1 Extrinsic MT Metrics 13
1.3.2 Intrinsic Tokenization Metrics 17
1.4 Charles Translator for Ukraine 19

2 Methodology **21**
2.1 Data 21
2.1.1 Datasets 21
2.1.2 Dataset Preparation 22
2.2 Pre-processing Algorithms 23
2.2.1 Casing 23
2.2.2 Diacritization 27
2.2.3 Romanization 33
2.3 Tokenizer Algorithms 35
2.3.1 SentencePiece 35
2.3.2 TokenMonster 38
2.4 Evaluation 40
2.4.1 Extrinsic Evaluation 40
2.4.2 Intrinsic Evaluation 44
2.5 Implementation Details 48

3 Issues in Stabilization and Normalization **50**
3.1 Data Normalization: Apostrophes and Inconsistent Casing 50
3.1.1 Inconsistencies with Apostrophes in Ukrainian data 50
3.1.2 Inconsistencies with Fully Upper-Cased Sentences 56
3.2 Stabilization of the Experiments 56

4 Experiments with Casing Strategies **61**
4.1 Main Inline Casing Algorithms 61
4.1.1 Intrinsic Analysis 65
4.1.2 Vocabulary statistics 70
4.2 Ablation Experiments 72

4.2.1 Disabling Fully Upper-Cased Strings 72
4.2.2 Enforcing Flag Separation by SentencePiece 73
4.2.3 Casing Augmentation 74
4.3 Intrinsic Comparison of Tokenizers Trained on Varying InCa Vocabulary Size 78

5 Experiments with Diacritization Strategies 82
5.1 Main Inline Diacritization Algorithms 82
5.1.1 Intrinsic Evaluation of the Czech Texts 85
5.1.2 Vocabulary Analysis 85
5.1.3 Analysis of Char-InDia Flags 86
5.1.4 Analysis of Hallucinations in Ukrainian-Czech MT Output 88
5.2 Ablation Experiments 88
5.2.1 Data Augmentation 88
5.2.2 Naive Char-InDia 90

6 Experiments with Romanization 92
6.1 Vocabulary Overlap Estimation 93

7 Comparison of Intrinsic and Extrinsic Metrics 96
7.1 Intrinsic Metrics Comparison 98

Conclusion 103
7.2 Ethical Statement 105

Bibliography 106

List of Figures 119

List of Tables 121

List of Abbreviations 129

A Appendix 130
A.1 General Statistics on Main Inline Casing Algorithms 130

Introduction

An atomic element of most of the natural language processing (NLP) tasks is a token. After decades of experimenting with tokenizing the texts by linguistic rules, regular expressions, or other heuristics, in recent years, the NLP community has switched to predominantly using the subword tokenization principle, which means splitting the texts based on the frequencies of the character sequences. This method allows us to encode the frequent words as single tokens and to split the rare words (that would otherwise have gone out of the restricted vocabulary) into smaller but frequent character sequences. Today, there are several popular approaches on which the subword tokenization can be based, for instance, BPE Sennrich et al. [2016] or SentencePiece Kudo and Richardson [2018]. Due to the unsupervised way of training and language agnosticism, subword tokenizers are used for the entire range of NLP tasks, including multilingual NLP and neural machine translation (NMT).

However, despite their obvious strong sides, subword tokenization algorithms have inherent vulnerabilities that are the reverse sides of their advantages. One of these problems is high sensitivity towards variation in the character usage, which is understandable (and sometimes even unnoticed) for a human. The examples of such variation are various types of casing (capitalization or upper-casing) of the words or omitting the diacritics where they are expected by the orthography rules. The subword tokenizers trained on the general-purpose data usually show poor performance on tokenizing the words which bear similar meaning but are written differently by casing or de-diacritization: since they have not seen enough training examples of different casing or diacritizations, they cannot find the corresponding lines for the upper-cased words and end up over-splitting it into smaller sequences they could find. The illustration of such over-splitting is shown in the table 1 below. Since tokens in the NMT (and other neural NLP models) are connected to their distributive representations, over-splitting of the word results in poorer performance on the final task; moreover, it increases the processing time as the input sequence gets longer.

This variation can be treated as noise and may be deleted beforehand, but in some cases it may also bear linguistic (as is in diacritics) or expressive (as in writing the sentences in all caps in the social networks) information. Thus, an ideal solution for handling such a variation would be to preserve the information about casing or diacritization while not damaging the quality of the tokenization. One of the solutions suggested for the casing problem and developed in a line of works is applying preprocessing on the texts before tokenizer. For casing, this means transforming the sentence to lower case, but preserving the information about the cased words via auxiliary symbols. The recent analysis by Jain et al. [2023] shows that, if applied with a number of tricks (for example, using a single auxiliary token for a sequence of uppercase words) and with data augmentation, it can handle the inputs well with different casings. However, the way the auxiliary symbols are assigned in this paper is questionable, as it allows both treating it as a separate token and merging the word with it (in this case, the transformation may lose its sense, since we aim to separate the character sequence from the casing information anyway). Moreover, most of the work on the inline casing

Variation	Input Phrase	Tokenized Sequence
None	Během výběrů	_Během _výběr ů
All-Caps	BĚHEM VÝBĚRŮ	_B Ě H EM _V Ý B Ě R Ů
No Diacritics	Behem vyberu	_Be hem _vy ber u

Table 1: An illustration of the tokenization problem with the phrase (in the second column) which was transformed either by upper-casing or by deleting the diacritization. For a general Czech speaker, all three phrases would be easily understood as the same phrase; however, the tokeniser struggles to split it consistently as it was trained on the "regular" data, where the words are diacritized and usually not upper-cased. In the third column, you see how the tokenizer splits the sequence into subwords (the splitting is represented by white spaces, the underscore is an auxiliary tokenizer sign denoting the actual white space). Contrary to the "normal" sentence in the first line, where the line is split into meaningful Czech words (or at least morphemes), in two lower lines it is over-tokenized into meaningless parts, which would influence the processing time and the performance on the downstream NLP task.

algorithms either focuses on the downstream performance of the tokenization on the NLP tasks or solely on the intrinsic performance of the tokenizers without its downstream NLP applications. Bearing in mind that subword tokenization is a relatively new technique and there is still no consensus in how to evaluate the efficiency of the tokenizer itself, this lack of simultaneous analysis of the intrinsic qualities of the tokenizer and the extrinsic performance of the systems which use this tokenizer is a big problem. Finally, to our knowledge, there has been no attempt to apply the inline approach to other orthographical transformations such as diacritics and de-diacritization mentioned above.

Another problem of the subword tokenizers is also rooted in its language agnosticism and concentration on the character sets used for training. For instance, two related languages that use different writing systems may have many common or similar words, but due to the mere fact that the character sets used to write their pronunciation are different, there would be no common representation of the similar words in the joint subword dictionary. Recent research on related languages, for example, Moosa et al. [2023], shows that another way of preprocessing can help. That is, mapping the texts in various writing systems to the common character set significantly improves the performance on the NLP tasks. Since our university is developing and improving the direct Ukrainian-Czech NMT system, we are facing exactly this problem, and we are interested in whether romanization of the Ukrainian texts (which are written in Cyrillic script) would help creating a more efficient tokenizer, as from the typological perspective, Czech and Ukrainian are considerably close. However, we have not seen a profound analysis of the impact of romanization of Ukrainian with respect to our translation pair.

Considering all the above, we decided to conduct an investigation of various pre-processing techniques with the focus on a particular NLP application - improving the performance of NMT for the Charles University Ukrainian-Czech translator. We are interested in analyzing and comparing various types of inline casing algorithms (including one suggested by us), developing an inline diacritization algorithm, and assessing the romanization of Ukrainian for the extrinsic task of Czech-Ukrainian and Ukrainian-Czech translation pairs and for the intrinsic metrics of the tokenizers.

The goals of our research are the following.

1. To suggest the inline casing solutions that improve robustness for different types of casing of the texts, without making the performance worse on the non-noised texts. Since we are not the first to propose this approach for the casing problem, our aim is to improve the stability and efficiency of our solution compared to the existing ones.

2. To expand the inline approach to the problem of diacritization and to try various principles of the auxiliary token preprocessing.

3. To apply the romanization of the Ukrainian for the task of direct Czech-Ukrainian and Ukrainian-Czech machine translation, and by that to improve the subword overlap of the two related languages without losing the performance on the downstream task.

4. To perform a simultaneous analysis and comparison of both intrinsic and extrinsic metrics (downstream, in our case, MT) of the tokenization. We are doing it in order to understand, firstly, which intrinsic metrics are informative for comparing the preprocessing methods. Secondly, our aim is to understand whether we can find reliable intrinsic predictors for the extrinsic performance of the NMT system which uses a particular tokenizer.

Our work is organized as follows. In Chapter 1, we provide an overview of the background in tokenization, its solutions with respect to noise, and its evaluation. In Chapter 2, we present the methodology of our research, explaining the data we use, the preprocessing and the tokenization algorithms we compare, and the metrics for evaluation of the algorithms' performance. Chapter 3 is related to the issues in text normalization and stabilization of our experiments. The three subsequent chapters, 4, 5 and 6, cover in detail our experiments in inline casing, inline diacritization, and romanization of the Ukrainian, respectively. Chapter 7 is dedicated to the comparative analysis of extrinsic and intrinsic metrics. In Conclusions 7.1, we summarize our results, show the limitations of our research, and outline the perspectives of our future work.

1. Theoretical Background

1.1 Tokenization: Main Approaches and Their Problems

1.1.1 Two Extremes: Words and Characters

The essential step in most natural language processing (NLP) applications is splitting the text into "atomic" units, usually called "tokens". These tokens are expected to bear the distributional information necessary for language modeling, be it n-gram or neural language models. Traditionally, a token was perceived as an equivalent of a "typographic" word (sequence of characters between white spaces or non-alphabetic symbols) or a "linguistic" word (whatever it is according to a particular language and a particular linguistic theory) [Mielke et al., 2021, p. 2]. Most of such tokenization packages, for example Moses Koehn et al. [2007] as one of the most popular, were deterministic and at least to some extent language and writing system specific.

However, a regular problem that arises during the inference of the language models is that a system can face an unknown token which was not seen during training [Jurafsky and Martin, 2023, pp. 20, 43]. This is called an out-of-vocabulary problem (hereafter OOV). This problem becomes especially serious when it comes to neural language models, as most of the neural NLP models (such as Bahdanau et al. [2014], Luong et al. [2015], Vaswani et al. [2017], Devlin et al. [2019]) work with fixed length vocabularies [Kudo, 2018, p. 66], where extending vocabulary ends up with higher computational costs. For this reason, using the "orthographic" or "linguistic" words as tokens is inevitably a suboptimal approach.

Another "extremal" idea that recently gained its popularity is to equate a token to a character[1] (with Chung et al. [2016] and Gupta et al. [2019] being among the first suggestions), or even to "zoom in" and base the language models on the byte level (with ByT5 model presented in Xue et al. [2022] being the most famous one). On the one hand, this approach solves the OOV problem as we can take into account all available UTF-8 symbols. On the other hand, it seems linguistically doubtful as, according to most language theories Haspelmath and Sims [2010], the smallest language element that has semantics is a morpheme. Even if we ignore that tenet, the major technical problem is increase of the computational complexities, as the length of such character or byte sequences for the neural models increases drastically. Libovický et al. [2022] show that the character and byte-level models do not show competitive results in the WMT competitions on

[1]The term "character" is used for Chinese and Japanese writing to denote the graphical symbols equal to syllabo-morphemic units. Thus, in the Chinese and Japanese NLP, the word "character-level segmentation/tokenization" was initially used for splitting the text into these characters, which obviously bear more semantics and phonetic information than the characters in the alphabetic writing systems. However, it is notable that Chinese characters can be split into graphical elements that are usually called "radicals" or "glyphs"; the characters can also be romanized. Thus, a corresponding level of the "European" character-level NLP in the Chinese and Japanese NLP is a subcharacter level, with the glyphs or the characters of the romanization as tokens, for instance, Meng et al. [2019] and Si et al. [2023].

the neural machine translation task (hereinafter NMT), and their only significant advantage is greater robustness of noise. The newer research, for instance, Edman et al. [2023], is more lenient towards the byte-level models (namely, ByT5), and demonstrates their advantage compared over the subword model of the similar architecture, mT5 (Xue et al. [2021]), in low-resource and cross-language transfer settings. Still, as of now, the opinion on character- and byte-level NLP models is far from consensus and there are many technical considerations (such as time inefficiency or UTF encoding inequality [Mielke et al., 2021, p. 14]) that prevent them from dominating in all domains and settings.

Before we move to the balanced solutions between the words and characters, we should also note that, for some writing systems, accessing the character (or sub-character) information seems more efficient than the coarse-grained tokenization. This is the case in Korean, where the basic unit of writing is a syllable which is a compound of 3 symbols (called "jamo") that represent the exact vowels or consonants. Cognetta et al. [2023], contrary to the general syllable-level approach, presents a recurrent neural architecture that uses the conditional information about the jamo that are generated for the current character, which allows to minimize the number of embedding parameters by over 99% while retaining the baseline performance on the tasks such as machine translation.

1.1.2 Subword Segmentation: Fully Unsupervised and Language-Aware

Finding the intermediate level of tokens between words and characters sounds like a promising compromise. This solution has been first introduced in Sennrich et al. [2016] as a byte pair encoding (hereafter BPE). This is a deterministic approach that is based on creating the dictionary of a given size, starting with the characters of the training texts and then iteratively merging two character sequences that are most frequently met in the text, until it hits the dictionary size. The resulting character sequences were called subwords. Such an approach was the first to solve both the OOV problem and the length of the processed string in terms of tokens, as, for the frequent words (or patterns of several characters), there will be one or very few corresponding tokens, while for the unseen words, the segmentation would be character-wise or containing character combinations (assuming every character is seen in the training data). Later, two other subword tokenization methods were introduced: WordPiece (the initial idea presented in Schuster and Nakajima [2012], developed in Wu et al. [2016]), and Unigram language model, better known as SentencePiece (approach presented in Kudo [2018], implemented in Kudo and Richardson [2018]). Contrary to BPE, the two latter models are based on probabilistic approaches; several approaches, BPE-dropout Provilkov et al. [2020] being the most popular one, introduced robustness to BPE.

Speaking of intermediate-level tokenization, we should also mention the experiments on the linguistically motivated approaches to retrieving morphemes of the words with different degrees of supervision – from supervised Gezmu and Nürnberger [2023] to mostly or completely unsupervised Morfessor family (Creutz and Lagus [2002] and later versions) or Eskander et al. [2020]. The comparison of the completely statistical and linguistically motivated subword segmenters shows that, for some languages, the morpheme segmenters can show better performance [Mielke et al., 2021, p. 11], especially when it comes to agglutinative languages (like Turkish) that use multiple grammatical morphemes for each word form. However, the same overview shows the opposite results for other languages. Overall, it seems that currently the statistics-based approaches for subword tokenizers are significantly more popular than the morphological ones, which may be explained by their complete language agnostic nature. For this reason, in the subsequent research we will pay more attention to the statistical approaches such as BPE or Unigram LM.

The neural approaches have become state-of-the-art solutions for most of the NLP applications within the last decades; especially in the last years, with the impressive performance of the so-called large language models (hereinafter – LLMs), which are mostly based on the Transformer architecture and are pre-trained on extremely large corpora (apart from aforementioned Vaswani et al. [2017] and Devlin et al. [2019] technological solutions, one of the first famous products was GPT-1 Radford et al. [2018]).

Since subword tokenization showed its efficiency on solving OOV problem and was also a convenient way of handling the trained vocabularies for the LLMs, the subword tokenizers became the default components of the neural NLP architectures. Thus, following the emerging convention both in academia Mielke et al. [2021] and in industry[2] we will use the term "**tokenization**" to denote subword tokenization unless specified otherwise, while the traditional approaches aiming word segmentation will be called "**pre-tokenization**". Correspondingly, the outputs of subword tokenization would be called "tokens" or "subwords", while the pre-tokenization outputs would be called "pre-tokens".

1.1.3 Inherent Problems of Subword Tokenizers

Despite preventing OOV problems, subword tokenizers faced several problems mainly related to excessive sensitivity to character variations. For instance, the default BPE or Unigram LM algorithm, when applied to the text in the orthography that uses cases for letters (e.g., English), will treat the same word in title case[3], lower case and upper case (for instance, "Hello", "hello" and "HELLO") as three completely different character sequences, ending up with different subwords, that would learn different vector representations after training. Specifically, the uppercased word will probably be segmented more heavily as it was either not seen at all or only a few times.

The same problem, dramatic change of the subword splitting due to minor typographical variation, happens if a tokenizer faces languages with diacritics (such as Czech), typos, or spelling variation in dialects or similar languages: for instance, Standard German "Schweiz" and Swiss German "Schwiz", both denoting "Switzerland", differ in one letter, will be split differently, and the corresponding subwords will be trained with no respect to the whole words' similarity. The problem is aggravated when it comes to similar languages with different writing systems (such as Arabic-based Urdu and Devanagari-based Hindi, or Turkic

[2]For instance, we can see the usage of the presented terminology in HuggingFace, one of the main hubs for neural architectures, datasets and pretrained models: `https://huggingface.co/docs/tokenizers/api/pre-tokenizers`.

[3]Hereinafter, the term "title-case" will be used for all occurrences of the first letter capitalization, not only the titles.

languages, which use Latin, Cyrillic, or Arabic-based alphabets): we understand that the number of potentially similar subwords on the phonological level should be large, but the small or zero overlap in characters will not cover it.

Tokenizers are also very sensitive to the handling of auxiliary symbols, for instance, on additional marking of the end of word (Macháček et al. [2018], up to 5 BLEU Papineni et al. [2001] points on German-Czech MT, depending on whether to mark the word boundary explicitly and how to handle it in the end of the sentence).

For several problems such as casing, a seductive solution would be to automatically augment the dictionary, for instance, create a BPE for lowercased words and then expand it by creating the corresponding titlecased and uppercased subwords. However, by this we violate the idea of minimizing the subword vocabulary size. This is aggravated by the fact that the performance of the NLP application depends heavily on the size of the subword vocabulary: the experiments show that the fixed MT architecture will vary by almost 10 BLEU points depending on the predefined vocabulary size Gowda and May [2020]. The problem of proper vocabulary size choice for the tokenizer became even more important with the increasing popularity of the massively multilingual LLMs (such as Conneau et al. [2020]), as it requires making decisions about the representation of various languages and various writing systems in the model.

Another solution to the problems of casing, diacritics, typos and other character variation (which we will hereinafter combine by the term "typographic variation") will be to preprocess the texts by minimizing this variation on training and inference steps. For instance, we could lowercase all texts, delete the diacritization, etc. However, this would obviously lead to loss of information that sometimes is grammatically relevant (for diacritics) or may contain stylistic or other subtle differences (such as writing fully uppercased or fully lowercased sentences in the social networks). Thus, the desired solution would be to come up with such a tokenization algorithm that would take into account the information about typographic variation for the same or similar words, but will also group the same or similar words closely in the subword space despite this variation.

For the reasons mentioned above, the NLP community started addressing the particular aspects of tokenizer sensitivity in order to improve their robustness. An overview of this field will be presented in the subsequent section.

1.2 Addressing the Typographic Variation by Tokenizers

1.2.1 Case Handling

In this section, we will cover the various types of typographic variations and the approaches that were suggested to solve them. We will start with casing information. One of the first solutions to it was provided in Rexline and Robert [2011].[4] The initial idea was to substitute the upper-cased and title-cased words

[4]Notably, the paper was initially intended for the purposes of text compression but not NLP. This is not a unique case of the compression algorithms inspiring the NLP solutions: the BPE algorithm also roots back to the compression algorithm solution Gage [1994]; moreover,

with their lower-cased correspondences, but to prepend them with the additional symbols that would mark the case of the word. This solution was implemented by the Naver lab for the MT task in 2019 Berard et al. [2019], where it was named "inline casing". Specifically, the additional symbols (we will call them and similar auxiliary elements "flags") were applied after the use of BPE to every subword that was upper-cased or title-cased; however, in this paper, according to the authors, it did not seem to have any significant impact. The next papers modified the approach. For instance, Etchegoyhen and Gete [2020] rather implemented the initial idea stated in Rexline and Robert paper and applied the casing flags before the BPE tokenization. Their results show that inline casing is the most efficient case marking strategy for several MT language pairs, compared to other approaches such as keeping the initial casing, lower-casing, true-casing, recasing and case factors (which will be described below). It showed both improvement in the overall BLEU scores, as well as a higher percentage of word reference matches with the cased words. The authors suggested that this happened because the inline casing is the strategy that allows "to combine lowercase-based translation benefits with case information exploitation" [Etchegoyhen and Gete, 2020, p. 3755]. Another study Shi et al. [2020] compared two variants of placing inline casing flags – either before or after the cased word, and trained the additional neural models for case prediction. They show that all approaches outperform baseline, specifically, the right allocation of flags works better than the left one. The authors suggested that the flags can help the system reduce the search space to translate the next word. However, compared to case prediction models, the inline casing showed a lower performance.

Another family of approaches for case handling is based on case factorization. In this scenario, the information about casing is disentangled from the word in the embedding step, after subword tokenization. For instance, Powalski and Stanislawek [2020] handled the case information in the same manner as the positional information in Transformer model – by embeddings that are added pointwise to the word embedding. This showed improvement on a variety of downstream tasks; moreover, a qualitative example showed that it especially improved the performance on the upper-cased words that were over-segmented otherwise. A more generalized approach was shown in Samuel and Øvrelid [2023]: the outputs of subword tokenization are transformed into 3-dimentional space by variational auto-encoder architecture Oord et al. [2017], resulting in a set of 3 integers in range [0, 255]. The results show that similar strings (differing not only with casing but also with other typographical variations) are represented similarly in such a format. The NLP task chosen to compare the performance was tagging on UD corpora De Marneffe et al. [2021], where it clearly outperformed the baseline BPE. It also showed significant robustness to character-level noise. Overall, this approach is a significant step forward in subword representation, as it solves the problem of non-relatedness of the similar subwords in all default tokenizer implementations. However, a drawback of this method is that VAE outputs are not as interpretable as the deterministic inline casing or similar approaches; they also

the recent observations of the GZIP text compression algorithms were claimed to be a good substitution for state-of-the-art solutions for text classification Jiang et al. [2023], however, this was soon disproved. The discussion took place on the GitHub page of the published paper: `https://github.com/bazingagin/npc_gzip/issues/3`.

need additional training for each new language.

There are other solutions similar to case factorization that also address typographical variations at the level of the neural model architecture. For example, Aguilar et al. [2021] combine the subword embeddings with the character representations, the latter being augmented by various operations such as random noise or case toggling for robustness purposes. This architecture is applied for several NLP tasks such as language identification or named entity recognition (NER), where it showed improvement and noise robustness.

1.2.2 Diacritics Handling

Another typographical variation aspect that we mentioned earlier is handling diacritics. Depending on the language, they can perform one of the two functions. On the one hand, they can be obligatory and bear phonological meaning, which is true for most Latin-based languages, from German and Czech to Vietnamese or pinyin, official romanization for standard Chinese. On the other hand, they can be perceived as auxiliary, which is usually true for consonantal alphabets such as Arabic or Hebrew, where the vowels are mostly marked with diacritics and are omitted everywhere except a closed range of domains (such as religious texts or books for children). Notably, the imporatance of proper handling of the diacritics in the current NMT is not only studied by the NLP community but is also analyzed by the formal and descriptive linguists: for example, Khoshafah and Tagaddeen [2023] in their qualitative analysis show that at least some of the widespread industrial MT applications (such as Reverso or Systran) do not take the information about vowel diacritics into account, thus not differentiating the translations of the diacritized and non-diacritized texts.

As for the NLP researchers addressing the problem of diacritics, there seems no significant body of research tackling the problem of encoding the diacritized letters: we have not found any serious discussion on handling the diacritization, comparison of diacritized and non-diacritized tokenization models, or different ways of diacritics normalization (representing diacritized letters as combinations of characters or as single characters). The default approach is to treat the letters with diacritics as the "atomic" characters in the same way as the "base" alphabetic symbols (which is mostly applied for the "obligatory" diacritics), or to strip the text off the diacritics (which seems to happen with consonantal systems and with some large multilingual models such as BERT Devlin et al. [2019]). The only brief mention and speculation about the impact of diacritics on vocabulary size is made in Alabi et al. [2020], where two low-resource African languages, Twi (which does not use diacritics) and Yoruba (which uses diacritics) are compared by their representations trained in FastText pre-trained models (usually of low quality) and on manually curated data. The authors speculate that, despite the fact that Yoruba orthography requires diacritics, there are not many properly diacritized open source data. Thus, the dilemma is either to use both diacritized and non-diacritized texts (which will expand the vocabulary) or to erase diacritics (which will lose phonological information).

There is an adjacent body of research related to the restoration of diacritics, both for languages with obligatory diacritics (such as the South Slavic languages Ljubešić et al. [2016] or Vietnamese Nga et al. [2019]), and for vowel signs for consonantal alphabets (such as Arabic Shamardan and Hifny [2023]). In most cases, the problem is formulated as an MT task from non-diacritized to diacritized language; thus, solutions such as sequence-to-sequence models or classical statistical MT architectures are applied to it.

1.2.3 Dialectal Variation Within the Same Writing System

The final significant aspect of typographical variation is the problem of dialectal variation or the use of similar languages. The approach suggested by Aepli and Sennrich [2022] suggests infusing character noise in the tokenizer training step, which showed an improvement in tokenization performance of closely related languages or non-standardized varieties of the standardized languages. This approach was replicated in Blaschke et al. [2023] and Srivastava and Chiang [2023], thus being proven on a wide variety of language families (from European languages like German and Norwegian to Indian languages and dialects of Arabic), and also showed improvement on downstream tasks such as PoS-tagging and sentence classification. As the latter paper shows, the resulting tokenizers start performing better even on the unseen varieties of the languages under question.

Another approach was demonstrated in work within the field of dialect classification Kanjirangat et al. [2023], where the authors tweaked the sizes of common vocabularies that would be minimal for a representative identification of a language variety. This approach is similar to the field of massively multilingual LLMs, whose first component is optimal choice of the subword vocabulary that would cover all languages in the training set: Rust et al. [2021], Chung et al. [2020]. Another related paper, Limisiewicz et al. [2023], will be analyzed more thoroughly in terms of the metrics suggested for tokenization evaluation.

1.2.4 Dialectal Variation with Different Writing Systems

The special case within dialectal and similar language variation is the phenomenon of languages and dialects that are very similar on the linguistic level but use different writing systems. Since one of the biggest regions with such dialectal and graphical variety is the Indian subcontinent, a large body of research is related to studying the unification of the writing systems before applying tokenizers. Usually, the final writing system is Latin script. In this case, romanization is applied according to a universal system for all Indian languages, such as the WX notation[5] in Kumar et al. [2023] and other unified packages such as in Moosa et al. [2023], or language-specific romanization schemes such as in Singh and Bansal [2021]. All demonstrated experiments show a significant improvement on the performance of both downstream tasks such as sentence classification or NMT, or on intrinsic parameters such as various metrics of subword splitting granularity; see [Moosa et al., 2023, p. 676].

A recent study that took into account a set of distant languages, Amrhein and Sennrich [2020], suggested applying different pre-tokenization romanization techniques to improve the performance in the NMT task. Of the two romanization packages used in the paper, uroman[6] assigned any input to the base Latin alpha-

[5]`https://github.com/irshadbhat/indic-wx-converter`

[6]`https://github.com/isi-nlp/uroman`

bet, making it more lossy but promised a larger character sequence overlap, while uconv[7] used a more elaborate system with diacritization, which allowed even full reversibility for some languages. This generalized analysis showed that romanization (which keeps as much detail as possible, uconv for this case) improves performance on the genealogically close languages that use different writing systems, while it does not have positive effect on the non-related languages. For some languages like Chinese (which did not have any related language in the sample), romanization showed a significant performance decrease.

The last paper that also tackles the initial writing system transformation (however, not within the scope of similar languages) was presented in Si et al. [2023]. The authors experimented with the effect of different encodings of Chinese characters on tokenization and downstream tasks. The characters were encoded either phonologically (using various approaches, including pinyin, standardized romanization for Chinese) or graphically (by encoding the strokes and radicals of the characters). Optionally, for the phonological romanization, the enumeration of homophonic characters was included. The results showed a significant improvement in intrinsic parameters, such as the speed of training and the coverage of the character combination (compared to Chinese character-level tokenization). The downstream metrics showed mixed results depending on a given NLP task, however, they showed robustness of the transliterations' encoding against random and homophone noise.

1.3 Tokenization Metrics for NMT

Since tokenization is the first step in any NLP pipeline, its efficiency can be measured by two essentially different approaches – either by the performance on a downstream task (which is called extrinsic evaluation), or by measuring how optimal the tokenization is compared to other tokenizers or to some ground truth parameters (intrinsic evaluation). Ideally, the intrinsic metrics should also be good predictors for the extrinsic metrics, as tokenization is a computationally-cheap operation compared to training the neural models such as a Transformer for an MT task. Thus, it would be fruitful to estimate the most promising tokenizer configurations before training and evaluating the final systems. Since our main focus is improving the MT task, we will be mostly interested in the extrinsic metrics for the MT, which will be presented in the next section, as well as cover the intrinsic metrics for tokenization.

1.3.1 Extrinsic MT Metrics

The general problem of MT quality evaluation is that, contrary to several other NLP tasks such as automated speech recognition (ASR) or part-of-speech tagging (PoS tagging), there is no single correct translation of a source sentence that we should expect from the system [Koehn, 2010, p. 217]. Because of that, the creation and adjustment of the metrics is a whole subfield in MT, with specific shared tasks related to that, for example, a yearly metrics shared task at the WMT conference[8]. In general, MT systems and MT metrics are developing in

[7]`https://linux.die.net/man/1/uconv`

[8]`https://wmt-metrics-task.github.io/`

parallel and contributing mutually.

Throughout the history of statistical and neural MT, an abundance of metrics have been proposed and used. We can try to classify them by several main parameters. We should start with the adequacy/fluency opposition: for the MT task, it is important to precisely transmit the information from source to the target (this is called adequacy) as well as to make the target sentence grammatical and coherent (fluency). The general MT metrics are aiming at covering both parameters, but for most of them, we can say whether a particular evaluation is more inclined towards adequacy or fluency.

The second dimension is the type of data that is shown to the metric: apart from the system translation, we can either provide an evaluation system with source sentence (source-based metrics, also known as Quality Estimation), a reference translation, usually made by human or by an oracle (reference-based metrics) or both (source+reference-based metrics).

The third dimension is the length of the system output that is being assessed: in most cases, it is the level of sentences, however, there are also document-based metrics (where the whole document is being assigned with a score) or document-aware metrics (where the assessor is provided with a document but assigns scores to the sentences).

The fourth differentiation parameter is whether the metric measures the surface features of the translation (e.g. character or word overlap), or its semantic contents.

The final parameter is the assessor itself, whether it is a human, a deterministic formula, or a stochastic (usually a neural) model. Below we will overview the main metrics by grouping them by this parameter, while briefly characterizing them by the above-mentioned dimensions.

Human MT Metrics

The first attempts to formalize human assessment can be found in the previous century, for example, White et al. [1994], where the authors addressed the features of adequacy and fluency directly. However, as shown in King [1996], most of the metrics were initially based on ranking the models or assigning points to them, and the assessors were not provided with rigorous differentiation criteria in the scores. The most widely used human metrics were designed in parallel with the classical automatic formulas. For example, the Human-targeted Translation Edit Rate (HTER), presented by Snover et al. [2006], was proposed together with an automatic metric of the Translation Edit Rate (TER). Both of these metrics can be described as source+reference sentence-based surface+semantic metrics. The metric principle is the following: how many edits are necessary to make (either manually for HTER or automatically for TER) to transform a system output to the reference sentence, which is not provided in advance but created on the basis of the MT output.

Another widespread family of metrics is called Multidimensional Quality Metric (MQM) Lommel et al. [2014], which is a framework that aims to identify the exact characteristics of the adequacy and fluency of the translation, and asks the assessors to score the systems with the points for each parameter. Depending on the exact set and granularity of parameters, the metric can be considered sentence- or document-based, surface- or semantics-based, but mostly it is source-based.

The third family of metrics, Direct Assessment (DA), was presented in Graham et al. [2013]: Contrary to the traditional ranking of the systems by humans, now the annotators were to provide a score for each translation. This metric is usually source-based, thus more semantics-based, and depending on the setup can be either sentence-based or document-aware.

Due to the obvious strengths of human evaluation (such as understanding semantics and possibly pragmatics of the text), human evaluation is the most valuable kind of metrics. On the other hand, it has several inherent problems that can be generalized by the term "bad reproducibility". This includes subjectivity of the annotators, not only based on the personal preferences on scoring candidates (which can be solved by normalization), but also based on their professional background. Freitag et al. [2021] showed that the judgments of the professional translators differ significantly from the crowd-sourced workers; moreover, Vojtěchová et al. [2019] noted that expert knowledge in the professional domain drastically affects human evaluation. Apart from that, there are other organizational problems related to the creation of the guidelines, inter-annotator agreement, and costs of human annotation. Thus, the creation of reliable automatic metrics becomes vital.

Non-Neural Automatic Metrics

The main advantage of automated metrics was supposed to be their speed, low cost, and reproducibility, whether to compare different systems or evaluate the progress of the same system. The most popular metric of this kind, BLEU Papineni et al. [2001], is a sentence-level reference-based metric whose core feature is counting the n-gram overlap with the reference translations (thus, the metric is surface based). The authors tried to combine the precision and recall constraints for the overlap, by applying the so-called n-gram precision and brevity penalty parameters in the metric. Introducing BLEU was a breakthrough in the field of MT (as well as other NLG problems), as it allowed to facilitate the research-and-development cycle given a relatively small and stable number of reference translations. However, since its very introduction, a large body of research has shown that BLEU has several significant drawbacks. Firstly, it is extremely sensitive to the number and quality of the reference translations. Secondly, since the metric is heavily surface-based by default, it is overly sensitive to preprocessing features such as tokenization Bojar et al. [2006], as well as to minimal (character or unigram-level) variations in the outputs. Finally, recent meta-analyses such as Ma et al. [2019], show that BLEU correlation with human judgements is uneven, reaching poor correlation with the top-scoring MT systems.

For this reason, several more robust modifications of BLEU were suggested, such as character n-gram F-score (chrF3) Popović [2015], which narrowed the BLEU concept of n-gram overlap to the character level. Another modification Stanojevic and Sima'an [2014] was based on a trained linear interpolation model with character-level or word-level factors, that allowed for more surface variation, such as gaps between the reference words. A family of METEOR metrics, presented in Banerjee and Lavie [2005], addressed several variation parameters such as morphological variants of the same lemmas.

The other automated approach, TER Snover et al. [2006], was based on the Levenshtein distance concept and evaluated the number of modification operations (such as insertion, deletion, substitution) to transform the MT output and the reference translation. This metric was still quite surface- and reference-based as BLEU and its modifications; however, combined with human assessment described in the previous section, it could allow for deeper structure and semantics of the outputs.

Finally, we should note that there was a family of approaches called quality estimation (QE), which are by default only source-based metrics. Progress in research and solutions in this field is more notable within the last years (see, for example, Specia et al. [2018]). The most popular approaches in the field have been QuEst by Specia et al. [2013] and Kepler et al. [2019]. The former QE framework is based on a combination of feature extraction (that try to comprehend both adequacy and fluency of translation) and classical machine learning, while the latter is based on neural approaches for prediction the human judgements based on sources and system translations.

Neural MT Metrics

The automated metrics based on deterministic algorithms, although they had significant imperfections and biases, allowed for quick and stable development of the MT systems. However, as can be seen from the inherent problems of reference-based (and surface-based) metrics and from the development of the QE metrics, the next logical step in the metric development was to rely more on semantics and less on the reference, probably with the help of deep neural networks. The first attempts to make trained metrics go back to the 2000s [Koehn, 2010, p. 244], but the boost in their popularity happened within the last few years and was related to LLMs.

The concept of embedding representations of the texts allowed one to first solve the inherent problem of the reference-based n-gram overlap metrics, the sensitivity to paraphrases; based on that, BERTScore Zhang et al. [2020] allowed for a more generalized and robust reference-based evaluation, which was demonstrated in MT and image captioning tasks. There were other experiments that were aimed at training a model that would work in low-resource MT, such as YiSi Lo [2019], or trained for greater domain robustness based on automatically generated data, such as BLEURT Sellam et al. [2020].

A new word in the MT evaluation was COMET Rei et al. [2020], the neural metric that was based on massively multilingual pre-trained LLM and could therefore make use of source and reference representations. It was generating the predictions on the human-assisted estimates and, according to the results, showed a significant correlation with human judgements on the top systems, which was usually a weak point for the previous metrics. Now COMET is a family of models that are trained on different types of input (some of the models are referenceless) or target variable (some models are aimed at ranking, while others predict DA, MQM, or HTER scores). As can be seen from our overview, the general feature of the neural metrics is their focus on semantic evaluation, while different models can be based on source, reference, or both.

According to recent meta-analyses such as Freitag et al. [2022], neural metrics are now more reliable compared to the classical n-gram ones. However, an even more abstractive meta-analysis Moghe et al. [2023] where the authors compared MT metrics (a variety of n-gram and neural ones) in the range of downstream tasks such as Dialogue State Tracking, shows that there is no clear correlation with metric types and downstream scores; however, reference-based metrics proved more informative compared to reference-free approaches (i.e., QE).

1.3.2 Intrinsic Tokenization Metrics

When it comes to evaluating the intrinsic tokenization, we can optimize several parameters of it. Firstly, an important feature of tokenization is reversibility of the encoded sequence, as a proof of lossless transformation of text. Sometimes we can allow for a small degree of non-reversible transformations: for instance, the first implementation of BPE did not guarantee the proper whitespace handling Kudo and Richardson [2018], while Amrhein and Sennrich [2020] discussed different ways of romanization with respect to how lossy it is and whether more lossy transformations could help generalization and robustness. However, in a general case we want to minimize the loss in the encoding-decoding cycle. The second objective is optimization in terms of text compression: we want to minimize the encoded length (number of tokens) given a fixed vocabulary size. The third parameter, which can be called the main priority of this work, is the robustness of the tokenizer against noise, new domains or vocabularies, and typographical variations. As was stated in the first section, it is quite problematic to optimize a tokenizer by all three dimensions, as improving robustness or compression usually leads to loss in reversibility, while maximizing reversibility minimizes the effects of compression.

Another concept that is not an objective but an opposition necessary to bear in mind is the opposition between frequency and compositionality capacities of the tokenizer. As Wolleb et al. [2023] show, when the main tokenization approaches, such as BPE and Unigram LM, were announced, they claimed that both features: frequency-based encoding of the subwords and efficient handling of the OOV (or very rare) words (which they call compositionality), contribute comparably to the state-of-the-art performance of the subword tokenization compared to other approaches. The authors attempted to disentangle these two features by creating a tokenizer that would only be able to handle the frequency problem and not address the OOV problem. They used another compression mechanism, Huffman coding, as a tokenization principle: by using highly multi-dimensional Huffman trees, they assigned the words with different IDs depending on their frequencies. They compared this approach to the standard implementation of BPE in the task of MT, and the algorithm showed a very high (around 90%) and consistent correlation with the performance of BPE. The authors' interpretation on this finding is that frequential aspect of subword encoding plays the main role in BPE efficiency, and urge the NLP community to think about the compositionality aspect of the subword encoders as of a task that is not automatically optimized by current tokenization approaches. We think that the metrics that will be described below can be described as being biased towards either the frequentist or the compositionality aspect of the subword encoding.

The research and formulation of the internal metrics for tokenization has started quite recently. It seems to be pushed forward especially by the multilingual LLMs, as one of the crucial problems there is which vocabulary length is optimal, and what is the optimal way to allocate the subwords from different languages and writing systems. For instance, Rust et al. [2021] state that the exact tokenization method is as important as vocabulary size for the downstream performance for the massively multilingual models. In line with this research, a recent master thesis by Jiří Balhar Balhar [2023] analyses the problem of the tokenizers' performance for the multilingual models. Balhar provides a thorough analysis of the internal tokenization metrics and compares them both from a theoretical and a practical perspective. In this overview, we will refer to this overview and generalize it in order to clusterize the possible metrics into several groups.

Metrics Related to Sequence Length

The first group of metrics addresses the length of the sequence encoded by the tokenizer. Our objective is to create the most coarse tokenization possible. It can be formulated differently – as average number of subwords per sentence Amrhein and Sennrich [2020] (also sequence length, Chung et al. [2020] and Liang et al. [2023]) or as average number of subwords per word (also known as fertility, Rust et al. [2021]). Balhar introduces a more generalized version of this metric, characters per token (CPT). He mathematically proves that the sequence length can be reduced to CPT with a corpus-dependent constant, and mentions that CPT is not as language-dependent as fertility, because the latter is based on the notion of the "word", which is not an intuitive concept for many languages or writing systems. Since this group of metrics is focused more on the lengths of each token or the resulting encoded sequence, we can say that it is rather compositionality-related as it describes the character length-subword correlation more directly than the frequency of the particular token.

Token Distribution Metrics

The second family of metrics addresses the distribution of the resulting subwords in a tokenized corpus. Sorting the subwords' occurrences by their frequencies allows us to create a variety of the probability- or entropy-related metrics. An intuitive idea is to analyze the parameters of the whole distribution: we want to have a distribution that would be closer to uniform, as this would increase the information of each token. For this task, metrics such as average log probability Zheng et al. [2021], entropy and average rank (AR) Balhar [2023] are introduced. Balhar shows that the suggested metric of average rank (which is the sum of token ranks weighted by their probabilities) is an optimal metric: firstly, average log probability is a combination of average rank and entropy; secondly, under the assumption that the tokens follow Zipfian distribution (which is usually the case for the natural language data), entropy correlates with AR, but is less sensitive to low-frequency tokens.

The initial idea of the subword tokenization is to handle the OOV words, which, in terms of Zipfian distribution, can be formulated as handling an (infinitely) long tail of the rare words. It is intuitively clear that we should pay special attention to the tail of the subword token distributions in order to make it as short as possible. Thus, several proposed distribution-based metrics specifically aim at the tail of the token distribution, such as Frequency at 95th% Class Rank Gowda and May [2020]: counting the least frequency in the 95th percentile of most frequent classes. Based on their analysis of tokenizers on the downstream MT tasks, the authors suggested that the optimal size of BPE is the largest size such that at least 95% of tokens would be met at least 100 times in the training corpus.

All the above-mentioned metrics (which we, by the way, can call frequency-related) were created ad-hoc and tried to measure the empirically obtained token distributions. Recently, a more fundamental metric was suggested by Zouhar et al. [2023]. It is based on the assumption that tokenization is a noiseless transformation and is based on the concept of efficiency, which aims at penalizing the token distribution on both head and tail. The metric is theoretically based on the notion of Rényi entropy, which is a generalization of Shannon entropy. The authors show that, on a variety of tokenizers and on a set of MT language pairs, this metric correlates well with the downstream external metrics such as BLEU. The metric will be presented in detail in the next chapter 2.4.2.

Comparison of Multiple Corpora

The metrics described above are concentrated on a particular tokenized text. There are a number of metrics that aim at evaluation of the similarity of two tokenizations for different corpora or languages (if it comes to multilingual tokenizers). A primitive metric of this kind is lexical overlap, which is an overlap of distinct tokens in the test data sets compared to the trained vocabulary (for example, used for the evaluation of dialect transfer in Srivastava and Chiang [2023]). We can see that it is a simplified version of the distribution-related metrics from the previous group, which does not weight the token occurrences and just counts them; however, the difference lies in its usage, as it aims at comparing the coverage of the training token set on multiple test corpora.

Another metric aimed at comparing multiple languages is suggested by Balhar: it is Jensen-Shannon Divergence (JSD). This metric is used to measure the distance between two distributions with the help of the midpoint distribution and the Kullback-Leibler divergence. This metric can be used to estimate the overlap between two languages given a multilingual tokenizer. Balhar also mentions that Chung et al. [2020] used Wasserstein's (or "earth mover's") distance for the same purpose, but he argues that using JSD is more justified since the earth mover's distance is defined for the probability distributions with metric space, while tokenizer vocabulary does not have one. In his work, Balhar analyzed the correlation between the intrinsic metrics and the downstream performance (which was implemented as a form of probing on a set on NLP tasks), and as a result, JSD as CPT showed a good correlation with the word-level downstream tasks (such as PoS-tagging).

1.4 Charles Translator for Ukraine

Since the main experimental contribution of this work is related to developing the Czech-Ukrainian MT system of Charles University, we have to briefly describe it. Charles Translator for Ukraine continues the development of the Charles University NMT solutions, which are in general based on the Transformer architecture Vaswani et al. [2017]. The distinctive features of this family of models (CUBBITT) are, first, block back-translation Popel et al. [2020]: the use of augmented backtranslated sentences separated from authentic bilingual data by different blocks. According to the authors, such a training approach allows for a better training curve than mixing the authentic and augmented sentence pairs within the same block. Another feature of CUBBITT systems that proved good performance in a number of shared tasks of WMT is called document-level translation Popel [2020]. In fact, it is a combination of a large attention context (which usually comprises several sentences) and an implementation of sliding context where the input sequences overlap for the sake of consistency throughout the text.

The CUBBITT version under question was created in March 2022, as a response to the massive inflow of Ukrainian refugees into the Czech Republic as a result of the Russian invasion into Ukraine[9]. It was presented at WMT-2022 Popel et al. [2022], where it participated in the Czech-Ukrainian translation task. In addition to architectural features, several language-specific elements that tackle data pre-processing were introduced. Firstly, the authors suggested the modified version of inline casing (InCa) that was frequency-based: instead of marking all upper- or title-cased words with flags, each word was analyzed by the frequency of its occurrences depending on case, and the default version was stored in the auxiliary vocabulary. Then, the most frequent type of occurrences was written as lower-cased and without any flag (even if it is upper-cased or title-cased), while the less frequent types were assigned with corresponding flags (including the lower-case flag for the rare occurrences of the lower-cased writings of words). This modification was aimed at generalizing the tokenizer vocabulary, on the one hand, and minimizing the encoded length, on the other.

Unfortunately, the authors did not have time to make an extensive comparative analysis of the InCa algorithm features. In this work we will thoroughly analyze the performance of various inline casing methods, as well as inline diacritization and romanization experiments for a clearer view of the importance of each factor in the Ukrainian-Czech MT system.

[9]The system is available for free at LINDAT repository: `https://lindat.cz/translation`

2. Methodology

In this chapter, we will introduce the methodology of the experiments on preprocessing of subword tokenization for the Czech-Ukrainian machine translation. This will include a general overview of the experimental setup in the following and a detailed description of each step in Sections 2.1–2.5.

The chapter covers the description of the training, validation and testing datasets (Section 2.1), the specification of the preprocessing (Section 2.2) and tokenizer configurations (Section 2.3) that we want to compare, and the evaluation approaches and metrics for the experiments (2.4). Finally, we will specify the technical implementation of our experiments in Section 2.5.

2.1 Data

2.1.1 Datasets

The problem of Czech-Ukrainian machine translation task stems partially from the fact that Ukrainian is currently a low-resource language even compared to Czech. There are active attempts to overcome this problem, for instance, within the Ukrainian NLP workshops Romanyshyn [2023]. Thus, we used almost all data from the open resources for the training procedure, and several small datasets (sometimes not open-sourced ones) for validation and testing.

For the training step, we used the same training data as was presented and described by Popel et al. [2022]: the dataset comprising 8 million sentences that contain all Czech-Ukrainian data from the OPUS corpus Tiedemann [2012], WikiMatrix data from the initial publication Schwenk et al. [2021], and the ELRC EU acts in Ukrainian.[1] The OPUS data consists of the multilingual web-crawled datasets that are usually met in the project, such as subtitles translations (OpenSubtitles, TED2020, QED datasets), localization files (Ubuntu, GNOME, KDE4 datasets), crowdsourced translations (Tatoeba dataset), Bible translations (bible-uedin dataset) and Wikipedia and CommonCrawl data (XLEnt, MultiCCAligned, wikimedia datasets, etc.). The data were preprocessed and filtered from malformed UTF-8 symbols, the sentences that were detected to be neither Czech nor Ukrainian, and the sentence pairs with too big difference in length. Finally, the handcrafted rules were applied to check the spelling of the named entities in two languages. The detailed explanation of the procedure is presented in the paper: [Popel et al., 2022, p. 353]. For brevity, the training dataset will be hereinafter called `T8M` (from "training – 8 million sentences").

For validation and various types of evaluation, the subset of 1012 sentences from Flores 101 dataset was used Goyal et al. [2022]. This is the dataset comprising the Wikipedia articles translated professionally to 101 language. Since the OPUS data partially comprised the Wikipedia data, we can name these data in-domain with respect to training data. For brevity, this dataset will be further referred to as `flo`.

The summary of the datasets is represented in the table 2.1.

[1] `https://elrc-share.eu/repository/browse/eu-acts-in-ukrainian/71205868ae7011ec9c1a00155d026706d86232eb1bba43b691bdb6e8a8ec3ccf/`

ID	Used For	Domains	Sentences	Words (cs)	Words (uk)
T8M	training	IT localization, web scraping, subtitles, Bible, tatoeba	7,905,591	82,019,407	81,601,852
flo	testing	Wikipedia (prof. translation) ∼in-domain	1,012	19,113	19,181

Table 2.1: Dataset Overview. The number of sentences and words was counted as the number of lines and the number of white space-delimited positive-length character sequences, respectively.

2.1.2 Dataset Preparation

We created the module for the controllable preparation of the datasets, which consists of the normalization module and the noising module. Both modules are described in the following.

Normalization

For the consistency and reproducibility of the experiments, we created a normalization module. It consists of several main aspects. Firstly, as we are working with languages that contain various writing systems and numerous diacritics, we should make the consistent formatting of the character sequences denoting the same "abstract" (in Unicode terms) character information. The Unicode normalization forms are used for that.[2] The normalization forms are distinct by two parameters. The first opposition is "canonical VS compatibility equivalence", which stand, respectively, for mapping various sequences of characters that denote the same abstract character to one representation or unifying the various visual variants of the same abstract character to one. The canonical normalizations (NFD, NFC) are mostly lossless and can be seen as one-to-one mappings, as they only unify the way the abstract characters are denoted; the compatibility normalizations (NFKD, NFKC) are many-to-one and then can be lossy. For example, for a diacritized Czech letter ř, it does not matter whether it is encoded as a single Unicode character U+0159 or as combination of "r" letter U+0072 and its diacritic "háček" sign U+02C7 (this is an example of canonical equivalence). However, the numero sign can be decomposed into general Latin "N" and "o" characters, but once decomposed, it will not be re-composed into numero sign back (this is an example of compatibility equivalence).

The other opposition tackles the treatment of the characters that consist of smaller parts (composites in Unicode terms): we can either consistently represent them as sequences of these parts (NFD, NFKD), or consistently combine them into single characters (NFC, NFKC). With an example of the Czech letter ř, the NFD and NFKD forms will decompose the letter into the "r" letter and "háček" diacritic, while the NFC and NFKC will unify the sequences of these two characters into one.

[2]`https://unicode.org/reports/tr15/`

Our data preparation module allows to normalize the text in any of the Unicode forms. Since our main interest is to compare the subword creation in different lossless settings, we will be primarily interested in the second opposition given the canonical transformation. Moreover, for our experiments, we consistently used the combining normalization. Thus, the normalization used in our setups was NFC.

Another parameter of normalization is handling the white spaces. The algorithms at many MT steps are sensitive to various types of white spaces; moreover, sometimes they distinct between single and multiple white spaces in the sentence. However, from the human reader's point of view, we can claim that these differences do not bear any "linguistic" meaning (or at least negligible compared to the "no VS single white space" difference). Thus, we decided to map the occurrences of multiple white spaces to single ones and to map all white space symbols except for a newline to the default white space.

Noising

Since one of our main aims is improving the subword tokenizers' robustness to noise, we included various types of noising of the input text. The first noising parameter is the casing. We can set the text fully upper-, lower- or title-cased. We can also set the percentage of the words that will be randomly cased (with upper-, lower- or title-case).

The second parameter is diacritization. From our experience, the main real-world noising with respect to diacritization in the Czech language is either full or partial omission of the diacritization. Thus, our noising system allows to either completely delete the diacritics from the sentence, or to delete it from a given percentage of words.

Note that in both types of noising, our system operates on the word level (for example, the whole word is de-diacritized or upper-cased). We do not infuse the noise that would tackle particular characters within the word.

Table 2.2 below shows the noising scenarios that we used for our evaluation in the work, namely, three scenarios of case noising (fully upper- or lower-casing, and 10% of randomly cased words), and of diacritization noising (full de-diacritization or stripping diacritics off 20% of the randomly chosen words).

2.2 Pre-processing Algorithms

The following section describes the crucial part of the work, the experiments with pre-processing (and post-processing) of the input texts for the Czech-Ukrainian MT. We will briefly motivate the ideas of our experiments, explain our proposed solutions and their variants, and mention similar solutions designed by other researchers that we compared in our work.

2.2.1 Casing

InCa – Inline Casing Algorithm

As shown in 1.2.1, one of the methods to handle the case variation is the inline casing. Its default idea is to prepend each word (in some approaches, the full

ID	Type	Short Description	Language	Aligned
none	case, diacr	no noise	cs, uk	+
upper	case	upper-case whole sentence	cs, uk	+
lower	case	lower-case whole sentence	cs, uk	+
$\text{rand}_{0.1}$	case	random case to 10% words	cs, uk	-
strip	diacr	delete diacritics from all words	cs	-
$\text{strip}_{0.2}$	diacr	delete diacritics from 10% words	cs	-

Table 2.2: Data preparation. The symbols mentioned in the "ID" column will be used to denote the text modifications in the ablation studies. The "Type" column shows which aspect of noising it refers to (either casing or diacritization); the "Language" column refers to the languages to which the noising applies; the "Aligned" column shows whether the operation is applied simultaneously to the pair of sentences or its application on the Czech and the Ukrainian sentence does not have to be aligned.

line or a subword) with a specific flag that would denote that this word is upper-cased or title-cased, and then lower-case the word. This approach, presented in Etchegoyhen and Gete [2020], solves the problem of inconsistency of subword extraction from the lower-cased and non-lower-cased character sequences. However, its problem is an increase (possibly significant if it is a long upper-cased sentence) of the input sequence.

Thus, Popel et al. [2022] suggested a solution that would combine the inline casing with optimization in terms of input line length, which was named InCa (from "INline CAsing"). The core idea is to collect the counts of each word in the training data about how frequently it was met in lower-, upper- and title-case, and to keep the information about the most frequent version of each word's casing in a vocabulary.[3]

Then, in the inference step, we access the vocabulary with each input word and check its most frequent casing. If the input word is cased the same way, it is lower-cased; otherwise it is lower-cased and prepended with a flag showing its casing. It is easy to notice that in this situation, we will need a third flag for lower-casing (for the case when the most frequent form is non-lower-cased, and the input word is in lower case), apart from the title- and upper-case flags. Since in both Czech and Ukrainian, the first words in the sentences are usually written with title case, the beginnings of the lines are treated separately: at the training step, the sentence initial words are lower-cased, thus do not add up to the counts of the title-cased variants. During the inference, they are checked against the vocabulary and are prepended with a title-case flag only if this title-case writing differs from the variant in the trained vocabulary (for instance, there it is upper-cased). Finally, we should handle the words unseen in the training step. In the inference step, they are treated in a straightforward way: they are explicitly marked with upper-case and title-case flags (with the exception of sentence initial

[3]For the purposes of vocabulary brevity, if the most common form of the word was lower-cased, it was omitted and treated as unseen.

position).

The post-processing part of the algorithm is responsible for decoding the text that contains the flags into the standard orthography. Each word is checked whether there is an explicit flag before it and, if so, it is used for the word casing. Otherwise, the algorithm queries the vocabulary, and, if there is an entry of the word, it returns its most frequent casing form. Otherwise, the word is left lower-cased.

There are several hyper-parameters of the model. Firstly, to filter out the infrequent words, we can set the minimal number of word occurrences to be listed in the vocabulary. Tuning this parameter can help minimizing the generalization error at the inference step. Secondly, since we "linearize" the information about the word casing and disentangle it from the typographical words, the position of the flag (whether it is to the left or to the right of the word) may play role in the performance of the algorithm, since the sequences of the tokens for the attention mechanism would differ. In this work, we compared only the left allocation of the flags. Third, for the sake of encoded length minimization, the initial InCa implementation featured a specific treatment of the all-uppercased sentences (for sentences of more than three words): instead of prepending each word with a flag, the sentence was prepended with a doubled uppercase flag; moreover, the words from these sentences were not counted in statistics, and thus did not influence the vocabulary. However, such a solution may decrease the quality of translation on the all-upper-cased texts as the network may have too few training data for such sentences. Thus, we can either keep this option on or turn it off for consistency.

The default parameters that are used in standard InCa implementation are setting minimal counts of cased words to 1 (i.e., taking into account all cased words seen in the training data), turning on special treatment of all-uppercased sentences, and left allocation of the casing flags.

Naive InCa

As was mentioned above, the main difference of InCa compared to other inline casing algorithms is the optimization of flag use by putting it only on unseen words or infrequent casings of the seen words. The hypothesis that this can help in the extrinsic MT quality step should be explicitly verified. To do that, we also applied the naive version of the InCa experiment, which differs from the main InCa in omitting the pre-trained vocabulary, thus marking each not fully lower-cased words with a particular flag, without regard to its counts. Logically, this can be seen as a limiting case of the main InCa, if the minimum number is put to a very large number that is never reached in the corpus; thus, each word is treated as unseen in training data. However, for the sake of time saving, the realization algorithm is not applied for training.

There are two parameters of the naive InCa to be tweaked: the first is the location of the flag, left or right, and the default value is left (same as in basic InCa). The second parameter is explicit or implicit handling of the sentence-initial words – whether we should mark all of them with explicit title-case flag or we can expect that by default and only mark the non-title-cased words with a lower-case flag specifically. In our experiments, we used the left allocation of the flags and explicit handling of the sentence initial words: i.e., every sentence-initial word was marked with title-case flag, contrary to standard InCa.

Marian Inline Casing

One of the recent implementations of inline casing algorithms was presented as a part of the Marian NMT project by Jain et al. [2023]. This approach aims to optimize the initial ideas of inline casing by Berard et al. [2019] and Etchegoyhen and Gete [2020], which introduced the flags for upper-, title- and mixed cases which are space-separated from the words (essentially analogous to the naive InCa approach). As the Marian authors rightly observed, such an approach was not optimal in terms of the tokenized sequence length. Their solution is as follows: first, each non-lowercased character in each word was prepended by a non-lowercase flag; then, with the finite state machines, the sequences of non-lowercased characters are grouped into either upper-cased, title-cased or mixed-case sequences; finally, if an upper-casing pattern spans over 3 words, it is marked by additional flags at the start and the end of this span.

Contrary to the naive InCa or the earlier approaches, the flags are not necessarily space-separated from the words they are marking. This is done in order for the SentencePiece algorithm to statistically decide in which words the token should not be separated (the more frequent sequences) and in which it should. The motivation behind this is similar to that of the basic InCa, since we are trying to minimize the explicit use of the flags by taking into account the casing frequencies of the words. However, it is based on the sequence transformation and does not require an external dictionary. Such a solution may look more elegant, but it has a feature that from the logical point of view is disputable: since we expect the frequent co-occurrences of the flags and (parts of) words to be merged, this would mean that the character-level representation of a particular frequent upper- or title-cased token is still not stripped off the casing information; it is just redistributed to the neighboring character. For instance, if we use the character T as the title case flag and we have the word "Prague" in the training corpus mostly title-cased, then after the preprocessing it would look like Tprague, and after SentencePiece traning this whole string would be assigned to a particular token ID. However, if we face a lower-cased or upper-cased words "prague" or "PRAGUE", they will not be mapped to the "Tprague" token as they are cased differently; this may only help in cases where the SentencePiece was trained to separate the flags from the words.

However, such a doubt is based on the theoretical analysis, while the paper in which the algorithm was presented shows increased robustness of the algorithm towards noised casing, as well as does not change the encoded length significantly on the noised data compared to the general text. It is necessary to note, though, that all algorithms (BPE without preprocessing, classical inline casing and the proposed algorithms) were showed better performance after using the augmented training data.

To assess that, we applied the in-built solution of the Marian inline casing preprocessor on our training data.

TokenMonster

The final approach that is covered within the inline casing is another outsourcing solution provided by TokenMonster.[4] This is a standalone subword tokenization

[4]`https://github.com/alasdairforsythe/tokenmonster`

system which claims to compress the text in an optimal and lossless manner, compared to other tokenizers. The details of its tokenization mechanism compared to BPE, SentencePiece and other subword tokenizers will be covered in Chapter 2.3; as for now, we should note that the algorithm allows for its own variant of inline casing (named capcode). Specifically, it introduces the auxiliary symbols W and C that are responsible for upper-casing the whole word or only the next character, respectively, while the input text is transformed to lower case. The approach is similar to the Marian inline casing by the fact that the capcodes need not be separate tokens and can be combined with other (usually auxiliary) symbols. The difference is that there are only two capcodes, for upper-casing and title-casing a single character, contrary to 4 flags in Marian inline casing system.

2.2.2 Diacritization

As described in the theoretical overview, we do not know about any approach that explicitly handled the diacritics for the tokenization task. However, inspired by the inline casing approach, we suggest a family of similar algorithms for the inline diacritization that we correspondingly call InDia (for INline DIAcritization). As a naming convention, we will call a unique sequence of Latin symbols that can take on various diacritics a "base". For example, the diacritized words "klícky", "kličky", and "klíčky" all have the same base, "klicky". [5] The general idea for all InDia algorithms is to collect statistics about various diacritics of each word base, encode various diacritizations with specific flags, and put such flags near the word while stripping off the diacritics from the word.

Leveraging the inline approach to diacritization may have two main extremes. On the one hand, we can look at the combinations of diacritizations in a word as at an atomic feature (for this reason, we will call this variant "**word-level** InDia" or, shorter, "Word-InDia"). For instance, if we have the base "hrabe" which may have diacritized forms "hrábě" (meaning rake) and "hrabě" (meaning count as a noble rank). If we do not pay attention to the number of diacritized characters in each word or to their characteristics, we can assign the first diacritization with a flag 1 and the second diacritization with a flag 2, which would become the flags for these words. Thus, the two words, "hrábě" and "hrabě", will be transformed to "1 hrabe" and "2 hrabe". Such an approach has the advantage

[5] Contrary to using inline approach to casing, there can be a general objection against it in diacritization. When we apply inline approach to casing, we are based on the assumption that all casing variants of a particular word actually denote the same "essential" word (for instance, in terms of semantics or lexicography), but are just represented differently in various contexts. However, when we address the diacritizations of the same base, it is obvious from the linguistic perspective that in many cases we are dealing with various lexemes, as for Czech the diacritized characters phonologically mean different sounds. For instance, three words, "zebra", "žebra" and "žebrá", have the same base, but map to three unrelated lemmas: "zebra" (zebra), "žebro" (a rib) and "žebrat" (to beg). For this reason, the whole idea of separating diacritics from bases may seem more doubtful. However, we can still provide arguments for such an approach, and the main reason is that, despite being non-standard, there is wide use of non-diacritized Czech in the Internet, which shows that in real life the non-related words are mapped to the same bases. We can also mention that there are corner cases of the inline casing, where the casing variants distinguish various lexemes, for instance, "US" (as the United States) and "us" as the indirect form of "we". However, the number of such lexical ambiguities with casing is orders of magnitude lower than the diacritization.

of encoded length, as it maps an arbitrary number of diacritized symbols to a single sign. Moreover, this approach is easy to be compatible with ordering the word diacritization by frequency, in a similar way to what was done with casing (the difference is that here we will have to put all variants of a bare word ordered by frequency in the vocabulary to keep track of the flags). However, such an approach has clear weaknesses. The main one is the lack of "semantics" in the resulting flags: since we only enumerate the diacritizations by frequency without regard to what these diacritizations are, the flags have no inherent "meaning" of a particular diacritization marking, contrary to the casing, where each flag stably denoted a particular orthographic operation. Here, in contrast, the flags 1 or 3 mean different operations for different words, thus potentially not leading the MT system to learn about the distributional properties of the embeddings of flags. Another problem with this approach to inline diacritization is handling the unseen diacritization: while for InCa unseen casing is explicitly marked with the same flags, here, we either have to keep the vocabulary updated even at the inference step, or leave the unseen diacritized words untouched as we have no explicit ID on how to transform them.

Another approach starts from the lower level of granularity: we can apply the flags to the diacritization operations on separate characters (we will call that "**character-level** InDia" or, shorter, "Char-InDia"). For example of "hrábě" and "hrabě", we can decide that a flag denotes the particular diacritizing of a letter; for instance, marking letter "a" with acute (in Czech – čárka) will be denoted with a flag 1, and marking letter "e" with caron (Czech – háček) is 3. Thus, the word "hrábě" will be transformed as "13 hrabe", while "hrabě" will be transformed to "3 hrabe". This approach brings back "semantics" to the flags, since each flag starts to denote the same orthographical effect throughout the whole text; it also solves the problem of unseen words for the same reason.

However, this comes at the cost of variable length of the flags for each word depending on the number of diacritized characters in it. Moreover, it is a tricky task to make a reversible but short transformation of a diacritized word into flags because if there are multiple characters that can take on the same diacritics sign and not all of them are diacritized with it, it is hard to associate the flag with a particular position without explicitly mentioning the location of the diacritic in the initial word. For instance, the base "radi" can be diacritized as "rádi" (meaning "happy [Plural, animate]") and "radí" ("[she/he] gives advice"), which both use the same "háček" diacritic but on different vowels. So, if a flag means putting "háček", it will not be clear which exact word was encoded with it. This can be solved by introducing different flags for putting "háček" on "a" and on "i" letters; but this will become problematic for words that have the same letters that are possibly diacritized, such as "pití" ("a drink").

We see that both approaches have complications that we can formulate with an inverse proportion: the more semantically interpretable flags, the less optimal in terms of space. Still, we may try to make the partial workarounds for both ways, that we will introduce below.

Before we get into detail with the description of InDia variants, we should note that in the Czech-Ukrainian language pair, Czech is more heavily diacritized, while in Ukrainian there are only two letters, ї and й, that use diacritics. For this reason, most of the discussion on diacritics handling will be referred to Czech side only (and will be applied only to the Czech data unless specifically noticed).

Word-InDia

As mentioned, this variation of InDia is the closest to InCa. It is organized as follows. In the training step, we retrieve the base of each word and make dictionaries of counts of all possible diacritizations of a particular base. Then, for each base, we sort the diacritizations by frequency and assign each diacritization with a flag.[6] We store all ordered variants of a particular base in the vocabulary, as, contrary to InCa, we cannot unambiguously retrieve the diacritization of a particular flag by itself, without an explicit mapping.

Then, at the encoding step, each word is checked against the pre-trained vocabulary. In case a particular diacritization of a base (or a base itself) is not found, in the default implementation the word is kept intact. Otherwise, each word is transformed into its base and a flag according to the vocabulary. The flags are always prepended to the word with the only exception: if the diacritization variant is the most frequent one, it is not explicitly marked with a flag. At the decoding step, if we see a flag with a subsequent (base) word, we look up the vocabulary and diacritize the base according to the flag. Otherwise, if a word is not preceded by a flag explicitly, we return the most frequent diacritization of the base from the vocabulary.

As was said, this version of InDia has problems of lack of flags' interpretability: for different bases, the same flag may mean different diacritizations. We attempt to partially solve it by allocating at least one specific flag with "stable" meaning: the one that denotes zero diacritics over the word.[7] The overall pipeline does not change, with only exception that now, the same flag is consistently assigned to non-diacritized forms in the vocabulary and the encoded data.

What are the parameters of the Word-InDia implementation? Firstly, we can make an implementation of InDia similar to the naive InCa, namely, order all diacritizations for each base, but mark the default one as the base form. This surely will not be optimal in terms of the length of preprocessed sequence; however, this may lead to some useful generalization. Thirdly, as with InCa, we can put the flags to the right and to the left of each word. Finally, we can treat the unseen diacritics with the seen base differently: the most straightforward way is to leave them as they are; however, for the sake of generalization (which comes at the cost of loss of reversibility), we can strip off the diacritics and assign the word with a specific flag with semantics of "unseen" diacritics. Another solution would be to dynamically increment the vocabulary at the inference step, being able to create new flags as we see the previously unseen diacritizations. However, this is tricky in terms of the real-world implementation, thus we did not try this scenario.

The default values of the above mentioned parameters are: ordering diacritizations from the most frequent one (not in the naive manner), allocation of the

[6] We use the circled digits in range 1-20, UTF range U+2460 – U+2473, in the descending order; however, the most frequent form is marked by the circled digit 0, U+24EA

[7] As a technical note, for the "equal to base" flag we use a specific flag – negative-circled zero sign, U+278A. However, this choice is made solely for the readability and otherwise any rounded number from the scope described above could be chosen.

flag to the right of the word, and no specific treating of the unseen diacritizations (leaving them untouched).

Char-InDia

A different approach to inline diacritization is based on perceiving characters as atomic units. We can map each diacritization operation to a particular flag, and, armed with that, de-diacritize the word and prepend the base with a sequence of the respective flags. Since the scope of our diacritization experiments is narrowed to a single language, the easiest variant of this approach will be to naively assign each occurrence of each diacritized words with the whole set of its diacritization flags. For instance, three words “zebra”, “žebra” and “žebrá”, given the operation flags 1 for putting háček and 2 for putting čárka, would be transformed to “zebra”, “1 zebra” and “12 zebra”, correspondingly. Since háček can be theoretically put on the “r” character, and čárka – on “e” character, such a mapping is too ambiguous. This can be solved in one of the two ways. Firstly, we can make a more granular mapping of flags by assigning each flag to each diacritic+letter combination. Thus, for háček transformation of the letter “z” there would be one flag and for “r” – another. This would narrow the ambiguity space but would not reduce it to zero, as we can imagine the cases with multiple occurrences of the same letter in one word. Another way is to specify the character position on which the diacritization took place. This can be done, on the one hand, by creating a sequence of flags equal to the length of the base, where each flag would correspond to a character (we will call that “linear” format). For instance, with word “žebrá” the annotation will be “10002 zebra”, with the 0 flag meaning there is no diacritization on this character. On the other hand, we can create position-flag pairs for each diacritization. For example, the word “žebrá” will be transformed to “1:1;5:2 zebra” showing that the flag 1 should be at position 1 and the flag 2 at position 5. This format can be called “key-value”.

Both formats have obvious drawbacks. Firstly, contrary to inline casing that generates a restricted set of flags with no combinations of them, here we should expect a combinatorial explosion in either of the formats, since the positions and the number of the diacritics would vary significantly across the corpus. Thus, while generalizing the bases, we will produce too many unique flag combinations. Speaking of each specific format, the “linear” format may guarantee more explicit diacritization patterns that can end up grouping into subwords. We can expect this due to regular morphemes and their combinations, such as the suffix of a verb nominalization -ání in the words like létání (flying) or vzdělávání (education). However, this format comes at the cost of doubling the pre-processed text length (in characters). The other, “key-value” format, should be significantly shorter as it would only point at the indices of the diacritized characters; however, it will lose the ability to generalize the subwords from multiple flags, as the absolute positions of the same morpheme or other diacritization pattern may differ.

Given all these problems, how justified is any attempt to implement any of the proposed word-InDia algorithms? To estimate that, we can look at the distributions of the frequencies of words in the training data according to their lengths, as well as the number of diacritized characters per word for each word length. The resulting statistics are shown in Figure 2.1.

The graph first shows that most of the words in the training data are not

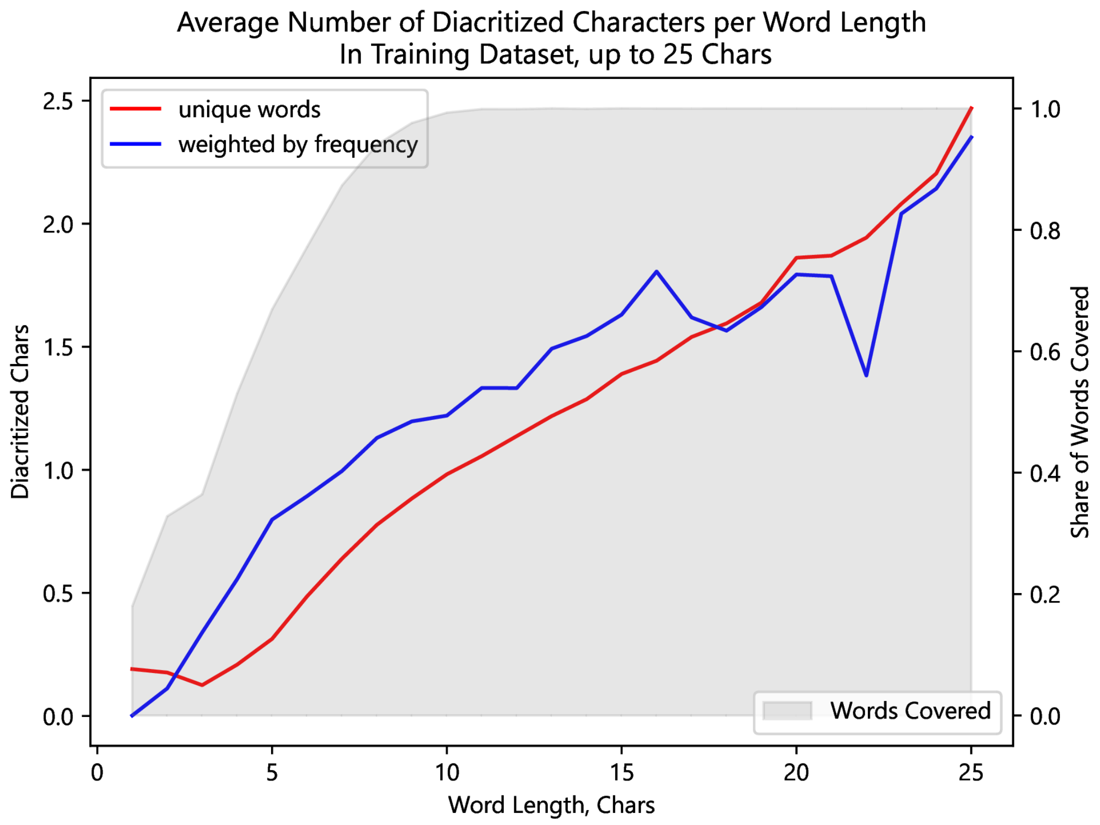

Figure 2.1: Average number of diacritized characters for each word length, calculated on the T8M training dataset (only Czech side). The x-axis represents the word length in characters (only subset of 1 to 25 characters is taken since it covers the majority of the words). The left y-axis shows the average number of diacritized characters for each word length; the two lines are showing the values (red is computed on unique word types, blue is weighted by the frequencies of each word type). The right y-axis shows the share of the cumulative distribution function of the words coverage in the dataset; the grey area is showing its values.

longer than 10 characters. Secondly, the average number of diacritized characters in the word is increasing slowly, roughly proportional to 0.1 of the length of the word. Although the average diacritized character rate is higher for shorter words, it is still not larger than 1.5 characters per 10-character word. This leads us to the suggestion that using the "key-value" format of diacritization flags is more feasible in terms of string length, since such sequences should be relatively short.

The problem left with "key-value" format is weaker ability to capture the flag patterns (in case they are multiple). We can approach this by reformatting the syntax of the flags. Suppose we have n key-value pairs: $k_1 - v_1, k_2 - v_2, ...k_n - v_n$, where k_i is a position of a diacritized character and v_i is diacritization flag.

We can store the pairs in the format $k_1 k_2 k_n | v_1 v_2 v_n$ instead of the more intuitive $k_1 v_1 | k_2 v_2 | k_n v_n$. Without losing full reversibility, we will group the diacritic flags on one side of our auxiliary word, which will leave space for generalization regardless of the absolute positions of the diacritics.

Can we minimize the lengths of the flag sequences even more? A possible solution can be to leverage the logic of frequency-ordered flags, such as in the standard InCa. We can store the most frequent diacritizations of each base in the pre-trained vocabulary, and mark with the flags only those diacritizations that are less frequent. This is an intuitive guess, but the statistics that we can collect from Word-InDia may support this claim. In the dictionary creation step, we sort the diacritizations of each base by frequency. Now, for each base, we can count two values: firstly, how distant (i.e. how many additional or different diacritics) each diacritized variant is from the most frequent form, secondly, how distant it is from the base (non-diacritized form). Since in the Word-InDia dictionary the diacritizations are ranked by frequency, we can evaluate the average diacritization distance of each rank in each of the two scenarios. We can do that with the Levenshtein distance metric Levenshtein [1966]. The result of this comparison is shown in the Figure 2.2

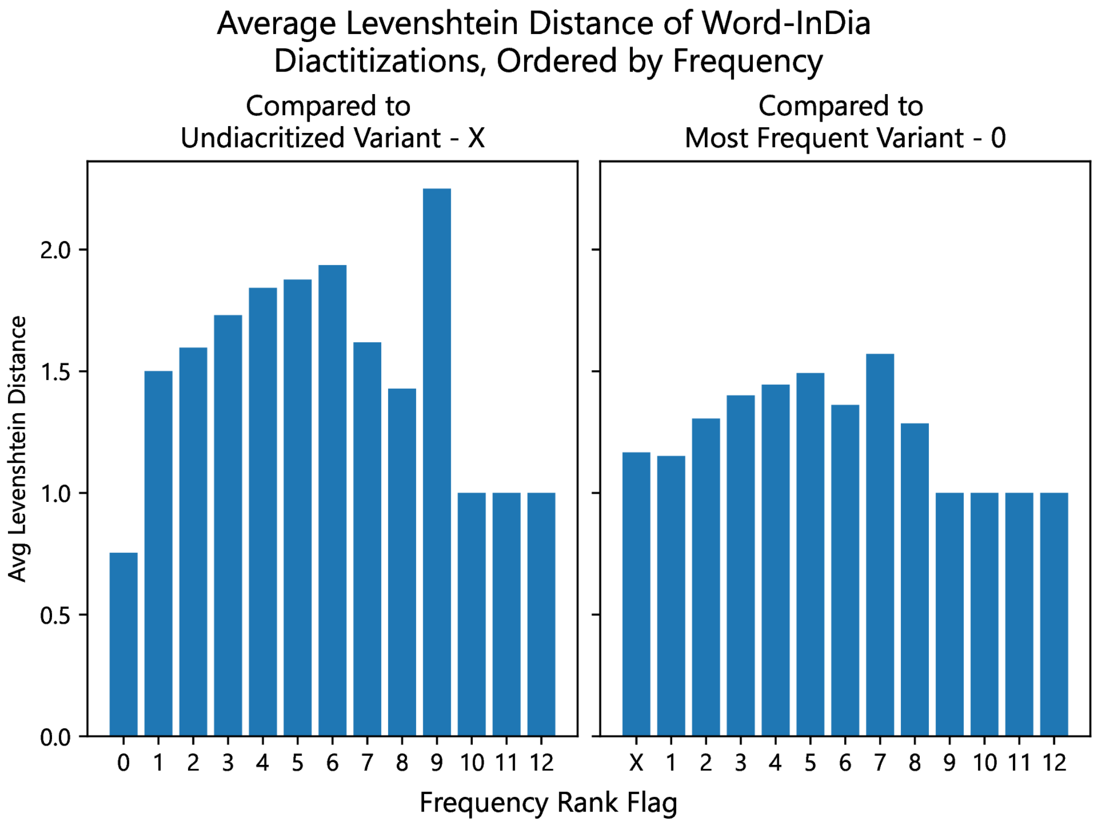

Figure 2.2: Average Levenshtein distance of the diacritization variants (ranked by frequency) for the Czech data. The x-axis represents the Word-InDia diacritization flags in ascending order. The number represents the frequency rank. On the right table, the "pivotal" words from which distance is computed is the base, thus all ranks (including most the most frequent, denoted by "0" flag) are shown. On the left table, the most frequent version is not presented as it is pivotal point itself; the negative "X" flag represents the base form in case it is present in the training data and different from the most frequent one. The y-axis represents the Levenshtein distance between each rank and the "pivot", which is averaged over the whole InDia vocabulary entries.

The table shows that the distribution of the ranked distances compared to the base has higher peaks and on average is approximately 1.5 characters, while if we measure the distances from the most frequent diacritization, the distribution becomes more uniform with an average of around 1.25 characters. This leads us to the suggestion that creating the pre-trained vocabulary of the most frequent diacritizations and marking only the deviations from them would be more optimal in terms of the encoding flag length.

Interestingly, this approach resonates with the way in which diacritics are used in a number of languages, especially in consonant-based writing systems. For instance, in standard registers of Hebrew and Arabic, the vowel diacritics are not expected to be written regularly, and one is expected to predict which vowel should stay after each consonant. However, if a writer thinks that a word's vocalization would be "unexpected" in the context (usually it happens with foreign proper names or ambiguous words), one can mark a full word with diacritics. Moreover, if only one syllable is opaque and other vowels meet the expectations of a reader, one can put a vocalization diacritic only on the position "under question", which is essentially our supposed way of diacritics of only the characters "diverging" from the most common diacritization.

The parameters that we can specify with this algorithm are, firstly, the left or right allocation of the flags. Secondly, we can choose the default (non-flagged) values to the base forms (this will be called the naive approach) or to the most frequent diacritizations of each base. The parameters that we used in the default implementation are left allocation of the flags, most frequent diacritization as the default form for marking divergent diacritics.

2.2.3 Romanization

Both languages in question, Czech and Ukrainian, belong to the same Slavic language family. Although they belong to different branches within it (West Slavic and East Slavic, respectively), the influence of the West Slavic languages on Ukrainian, especially Polish, has been intensive throughout the centuries. Thus, we can expect that in spoken mode of languages, the lexical overlap between the two languages should be substantial. This may help a lot at the tokenization level, as the joint vocabulary trained on both languages can have a substantial number of tokens used by both languages, not only by one of the two.

However, there is a significant obstacle on the way to that: Czech uses the Latin alphabet, and Ukrainian uses Cyrillics. Thus, despite many words being the same on the phonetical and phonological level (for instance, Ukrainian "слово" and Czech "slovo" meaning "word" are pronounced in the same way as [slovo]), or at least differing only in one or several sounds (such as "урода" and "úroda" meaning "harvest" with the only difference in vowel length), the overlap of the orthographical words would equal to zero just because of various character sets.

The logical solution is to map one of the languages to the writing system of another one. In our case, it would be the romanization of Ukrainian. There have been various standard or widespread approaches to romanizing Ukrainian texts throughout history and today. For instance, in the XIX century there were competing versions of the Ukrainian romanization, for example, by Josef Jireček, which was similar to Czech by using diacritization signs, or by Josyp Lozynski, which was sticking to Polish orthography in terms of using digraphs. Such systems did not provide a simple one-to-one Cyrillic-to-Latin mapping, but also suggested various framings of the Ukrainian orthography rules; thus, such approaches are not perfect for our technical transformation of the Ukrainian to the

Latin base as an intermediate preprocessing step. Currently, there are latinization approaches to Ukrainian, such as the national standard approved in 2010 by the Ukrainian Cabinet of Ministers, as well as the more recent standards such as DSTU 9112:2021. Some of such standards even have packages for automatic romanization and deromanization texts.[8] However, all these standards have at least one of the two problems: firstly, they may suffer from the non-reversibility (for instance, in the 2010 national standard both letter and "йе" combination may be mapped to "ie"). Secondly, Ukrainian phonemes that overlap with Czech may be encoded differently from Czech (for example, Ukrainian letter "г" in the DSTU standard is substituted either by a "g" with breve or by "gh" symbols, while in Czech the same phoneme is denoted by "h"). If we used that, we could have significantly decreased the overlapping vocabulary even within the scope of the Latin script, despite our main task being to increase it as much as possible.

Bearing these two factors in mind, we decided to use the romanization created by Martin Popel for the WMT-22 Czech-Ukrainian shared task Popel et al. [2022]. It was used within one of the ablation experiments but did not show any significant improvement and thus was not covered in detail. The main principle of this algorithm is based on, firstly, full reversibility, secondly, biggest character overlap of the romanized Ukrainian and Czech texts. For most of the characters, it works as a form of one-to-one mapping of a Ukrainian character to its Czech correspondent. The main difference lies in the scope of treatment of soft (or palatalized) consonants. According to Ukrainian orthography, the information about consonant softening is contained not in the consonant itself but in its right context, either in the iotized vowel or on the special letter "soft sign", which "softens" the preceding character. In contrast to that, in most cases for Czech, the palatalization is marked on the palatalized consonant itself. For example, the palatalized consonant [r^j] in the Ukrainian word "ріка" is encoded in the letter "і", which stands after the letter denoting "r" consonant. At the same time, the analogous palatalized sound of its Czech cognate "řeka" is contained as a diacritization of the "r" consonant itself. In our algorithm, a set of bigram character rules is applied in order to relocate the palatalization signs to the consonants where it is possible, and in other cases to insert the Czech letter "j" denoting the iotation.[9]

The initial romanization algorithm left one character of the Ukrainian alphabet that does not have a correspondence in Czech untouched – it is the soft sign (ь). In Ukrainian, it is used for the palatalization of the previous consonant and as a part of the iotized sound "o" (resulting in the digraph "ьо"). Where possible, this sign's information was redirected to the palatalized Latin consonant; however, for non-dental consonants it is not possible within the Czech orthography. In such cases, the soft sign was used as is in the romanized version of the text. It appeared to be a problem for tokenization, as the default implementation of the SentencePiece tokenizer splits all occurrences of different Unicode scripts, thus it would consistently dissect all occurrences of the soft sign within a word (a detailed discussion on this can be found in 2.3 and in 3.1.1 with a rela-

[8]For instance, 13 romanization standards implementation: `https://github.com/dchaplinsky/translit-ua`.

[9]Another detail, handling the Latin sequences which are already present in the Ukrainian, is handled by marking the spans within which the characters are left untouched after the backward cyrillization.

ID	Type	Short Description	Params and Variants
base	casing, diacr	no preprocessing	
inca	casing	InCa algorithm	`m` - min count, `n` - naive, +a - incl. all-caps in statistics
tkm	casing	tokenmonster capcodes	
marian	casing	marian inline casing	
india	diacr	InDia algorithm	type - `char` or `word`, `n` - naive, scope - `cs` or all
roman	casing, diacr	romanization	+soft - encode soft sign as Latin

Table 2.3: Overview of the pre-processing algorithms. The names in the "ID" column and the parameter labels in the "Params and Variants" column will be consistently used hereinafter to refer to the experiment configurations. The "Type" column describes whether this preprocessing is applied for the casing or diacritization experiments.

tion to another problem, apostrophe sign). Thus, we created a modified version of the romanization, where we used a symbol that belongs to the Latin script part of the Unicode table, so that splitting by the soft sign is not enforced. We compare both romanization methods in combination with the standard InCa and char-InDia no-prep-rocessing approaches.

2.3 Tokenizer Algorithms

We are using two tokenizer algorithms in our research, which will be described in detail below. For the sake of consistency, in all setups we train a joint subword vocabulary on T8M data, and the size of the vocabulary is 32000 tokens.

2.3.1 SentencePiece

The main tokenization algorithm used in the work is the implementation of the Unigram LM in SentencePiece, presented in Kudo and Richardson [2018].[10] It has been one of the dominant solutions for subword tokenization in recent years, together with BPE Sennrich et al. [2016] and algorithms based on WordPiece Schuster and Nakajima [2012], such as tokenization in BERT Devlin et al. [2019]. The main difference between BPE and WordPiece, on one hand, and Unigram LM, on the other, is that the formers are the bottom-up algorithms that start with single characters and gradually merge either the most co-occurring pairs of the subwords (as in BPE) or such pairs that maximize the n-gram LM probability of the segmented text (as in WordPiece). SentencePiece, on the other hand, is a

[10]The implementation is the fork of the main sentencepice library in the Marian repository: `https://github.com/marian-nmt/sentencepiece`.

top-down algorithm that starts with creating a large dictionary of the frequent words and common substrings, and then, with the help of the EM algorithm, filters out the subword candidates that maximize the loss of the unigram LM for this corpus. When the pool of subword candidates is reduced to the desired number, the training stops.

Why is SentencePiece chosen as the main tokenizer in our work and why is it popular, in general? The Unigram LM approach is a solution to an important vulnerability of the greedy subword algorithms such as BPE, as usually the same word can be tokenized into various subword combinations, and the greedy algorithms such as BPE do not take into account the probability of the resulting subword sequence. There were several solutions to improving the robustness of BPE toward ambiguous sub-word splitting, for example, Provilkov et al. [2020], however, it seems not to have become a popular solution compared to basic BPE or Unigram LMs. As for the statistically-based WordPiece, contrary to that solution, SentencePiece's implementation of the top-down approach has a less complex architecture, since it only takes into account the 1-gram level language models. [11]

The SentencePiece implementation of Unigram LM is a full-fledged package accessible in various programming languages that provides a wide range of training and encoding parameters that a user can specify. Some of these parameters are important for our research. Firstly, it is input_sentence_size parameter that is responsible for taking the subset of the input data to train the tokenizer; in all our experiments, this parameter is equal to the total of all sentences in Czech and Ukrainian T8M data[12]. The only exception is the augmentation scenario, where the value of our parameter is the same, while the size of the training data is increased by 4, thus, only a quarter of the augmented training sentences is used for training the tokenizer.

The second group of parameters tackles a distinguishing feature of SentencePiece implementation (of both its Unigram LM and BPE): it encodes the white spaces as specific symbols, since this allows the tokenizer to distinguish between word-internal and word-boundary character sequences. Such a distinction was perceived efficient in the earlier literature (for instance, in Macháček et al. [2018] for specifically the task of NMT) and is now a widely-used practice. It is worth noting that recent research for the encoder-only models in non-NLG tasks (such as NER or classification) shows that at least WordPiece subword vocabularies do not lose in performance if the information about the spaces is fully omitted; moreover, the tokenization of multimorphemic words becomes more

[11]It is worth noting, however, that the latest research Beinborn and Pinter [2023] on comparison of the human cognitive plausibility of subword segmentation shows that human judgement on plausible word segmentation correlates more with bottom-up approaches such as WordPiece and BPE. This is an important note in the perspective of general interpretability of the subword segmentation and NLP solutions; however, we leave these arguments beyond the scope of our work since we are aiming at the robustness and the MT system performance of the tokenizers and preprocessing algorithms.

[12]Unfortunately, even the explicit specification of the number of training lines does not guarantee leveraging all data in training, since it is computationally costly. Because of that, the algorithm subsamples approximately 5M sentences from a large training dataset (which are in our case approximately 16M sentences). However, the sentence sampling algorithm seems deterministic or controlled by seed, since rerunning the training on the same datasets leads to the same subword vocabularies.

aligned with morphology. Still, such findings are restricted by both the scope of tasks, languages, and tokenizer architecture, so we adhere to the common practice of marking the whitespaces of specific affixes to the words. The parameter treat_whitespace_as_suffix is responsible for choosing whether the whitespace affix should be before or after the word; the default value of the parameter is false and the spaces are treated as prefixes. For most of the experiments, we follow the default value of prefix marking of the white spaces; the only exception is experimental setup with Marian's SentencePiece inline casing algorithm described in 2.2.1.

The two other parameters are related to the restrictions of the subword search space. The parameter split_by_whitespace defines whether we restrict the subwords to go beyond the word boundaries (which are defined by white space, beginning and ending of the lines, and non-alphanumeric sequences). If the parameter is set to False, the subwords can span over multiple words. We set the parameter to True, the default value (contrary to TokenMonster algorithm that will be described thereafter in 2.3.2). Another parameter, split_by_unicode_script, defines whether a subword can contain the characters of the different script classes from the Unicode systems (for our work the important classes are Latin including numerals and punctuation, Cyrillic, "Code for undetermined script" which includes punctuation and diacritization, and Vai syllabary which is used for InCa and InDia flags). The default value of this parameter is True, thus the characters of different scripts are not to be placed together. However, when we go beyond the scope of the "Standard Average European" Latin-based languages, we will see that clustering of Unicode symbols into scripts becomes less intuitive and sometimes completely unexpected. One of such examples, the Ukrainian apostrophe, will be covered in detail in Section 3.1.1; In this section, we tried to analyze whether disabling the unicode splitting parameter brings more advantages or disadvantages in tokenization. With the exception of the experiments in apostrophe normalization, for all other experiments, we used the default value of splitting the characters by Unicode scripts.

The next parameters tackle handling of the characters that we plan to use as flags. The user_defined_symbols parameter lets the user define symbols that would be treated as a separate subword in any context. We do not use these parameters by default since our preprocessing algorithms already ensure that the flags are separated from the alphanumeric sequences. The only exception is one of the ablation experiments 4.2.2, where we compare the performance of the MT system with the standard InCa with and without specifying the flags for SentencePiece.

Another parameter which is important for flag handling is required_chars. The SentencePiece training process starts with defining "alphabet", a set of Unicode characters from which any subword in the vocabulary would consist. Since the training corpus may have single or very rare occurrences of some characters, the required character coverage is set to 99.95% of the characters seen in the training dataset that were observed most frequently (this is a default value of a character_coverage parameter which is also adjustable). However, on top of that, we can manually add characters that we will require to be included into subwords disregarding how rare they are seen: this is a task of the required_chars parameter, and we had to use it in the InDia experiments, since our flags were by default neglected by the SentencePiece trainer. Finally, normalization_rule_name parameter is responsible for UTF normalization before training and inference of the tokenizer. It only allows for 2 options: NFKC normalization (as the default value) and "identity", i.e. preserving the current format. For most of the cases, NFKC normalization is permissible for our tasks as the general alphanumeric sequences in our data do not change under this process. However, the flags that we are using for InDia are vulnerable to that, thus we use "identity" normalization for InDia experiments and NFKC normalization otherwise.

2.3.2 TokenMonster

TokenMonster is a new tokenizer whose stated objective is to optimize the endoded token lengths and the encoding time at the inference stage. Not only the algorithm, but also a number of pre-trained models are available online at its Github repository[13], as well as a web interface showing comparison between the state-of-the-art tokenizers used by LLMs and this system. The creator shows that with TokenMonster, "text can be represented with 37.5% fewer tokens at the same vocabulary size compared to other modern tokenizing methods", which means the solutions such as GPT-2 tokenizer or Tiktokenizer[14] that are used in OpenAI models. The author also states that, compared to these algorithms, "vocabulary size can be reduced by 75% or more".

Unfortunately, there is no available publication or detailed documentation of the TokenMonster; we can judge about the method based on the description in the Github repository. According to it, the concept of TokenMonster is similar to top-down n-gram language model approaches such as the one presented in SentencePiece, although the author does not mention this term and calls his approach "distillation". Firstly, the algorithm generates all possible tokens based on the corpus and deletes such tokens that occur in the corpus less than threshold times. Then, multiple vocabularies of the pre-set vocabulary size are randomly sampled. Each vocabulary tokenizes the text, and 1% of the "worst scoring" tokens is deleted (the scoring technique is not explicitly provided). Then, when the vocabulary size is significantly reduced, the previously deleted tokens are resurrected, and the process is repeated multiple times. The iterative process stops when there is no improvement in tokenization quality after 1000 iterations.

An important parameter in the training process is an "optimization mode", which defines how strict the requirements are to allow a sequence to be a token. There are five optimization modes ordered by strictness, and the author announces that the strictness of the modes correlates negatively with the average length of the tokens in the text. Since the tokeniser is planned not only for natural language applications but also for code and for bioniformatics, some modes are advised only for non-NLP use. Bearing that in mind, the recommended modes for the usage in NLP task are two strictest modes – "consistent" (second strictest) and "strict" (the strictest). The "strict" mode enforces encoding the alphanumeric sequences in the same way as much as possible (from the example on the website, the sequences "however", "However" and "However!" will be represented using the same "however" token with the specific tokens for casing and punctuation where

[13]https://github.com/alasdairforsythe/tokenmonster

[14]https://github.com/dqbd/tiktokenizer

necessary). The “consistent” mode allows more variation in this respect but still limits the delimiters and other sequences for a bigger consistency of the word and multiword expression collocations’ representation.

From what we can see, the difference between the SentencePiece implementation of the Unigram LM tokenizer and TokenMonster is the way the subword filtering process is organized. While in SentencePiece, only one vocabulary is generated and then the words with the smallest loss are iteratively filtered, TokenMonster first parallelizes the vocabularies sampling, and secondly iteratively returns the previously filtered-out tokens. The latter feature allows the author to call the algorithm “non-greedy” and to point it as a positive distinction from BPE; although this is true, the WordPiece and SentencePiece approaches, which can hardly be called greedy because of their probabilistic nature, are not mentioned and compared on the website. Finally, there is no specification of the loss function that is used to define the “worst” tokens; it may also differ from the LM likelihood in the SentencePiece approach. Unfortunately, we cannot comment on these differences in a detailed manner because of the lack of documentation.

There are several parameters of the tokenizer that a user can regulate before training. Some of them were crucial for our task. Firstly, it is choosing the optimization mode described above. For our experiments, we followed the advice of the author and performed two experiments – one with the “strict” mode and another with “consistent” mode. They showed similar results; therefore, hereinafter we will only refer to the “strict” version of TokenMonster. Secondly, TokenMonster allows us to choose whether we want to use the “capcodes” (the auxiliary tokens are used in the same way as the InCa flags, described in detail in Section 2.2.1). In both experiments, we use this option and use such capcodes. Third, the “norm” parameter allows normalization of the input data. The default approach is NFD Unicode normalization; however, since it is different from our basic NFC approach, we disable this function and leave the data as intact as possible (it is not possible to choose zero normalization, thus we use the “unixlines” value which is supposed to change the Microsoft newline combinations to UNIX ones, which we preliminarily do for all experiments). It is necessary to note that all these parameters are applied for the first step of the algorithm training – possible token extraction; all parameters for the iterative training process are left default, except for specifying the resulting vocabulary size, which is 32,000.

It is important to remember that TokenMonster by default takes into account the tokens that span over multiple words. This is obviously one of the features that allows it to increase the character per token ratio and decrease the encoded lengths, as many frequent multi-word expressions in a natural language can be mapped to a single token. What makes it different from most of the other tokenizer systems is that TokenMonster does not allow disabling this function. We must bear this in mind when comparing the intrinsic metrics of TokenMonster with those of SentencePiece.

2.4 Evaluation

2.4.1 Extrinsic Evaluation

Since our focus is machine translation, we used three metrics specific for the MT performance evaluation that are being widely used in the community throughout the last few years. Two of them, BLEU Papineni et al. [2001] and Popović [2015], are surface-based, reference-based and n-gram based (see classification of these parameters in 1.3.1). The third metric, COMET Rei et al. [2020], is a neural, semantics-based and source+reference-based metric. These metrics will be presented in more detail below, and Table 2.4 will provide the abbreviations that will be used in the experiments and discussion chapters.

BLEU and Its Lower-Cased Version

The principle of the BLEU metric is to compute the number of n-gram overlaps between the system output and the reference translation(s). This is done by computing the weighted average of the n-gram count matches for various n (usually 1 to 4). Since a naive ratio of correctly translated n-grams over all generated n-grams can be abused in several ways: either by generating as many various tokens as possible so that recall is inflated; or by guessing a minimum subset of words and stopping in order not to increase the denominator. To handle such tricks, two features are introduced. The modified n-gram precision takes into account how frequent a particular candidate n-gram was met in a particular reference; and brevity penalty is used to encourage the system to not output the sequences which are substantially shorter than reference (based on average ratio of the system output to reference sentences). The final metric is formalized in the following formula:

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.1}$$

where:

- **BP** is the brevity penalty, counted as

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

  where:

$$c = \text{length of candidate translation}$$
$$r = \text{length of reference translation}$$

- N is the maximum n-gram order considered,
- p_n is the modified precision for n-grams, counted as

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \{Candidates\}} \sum_{\text{n-gram}' \in C'} \text{Count}(\text{n-gram}')}$$

, where:

$$\text{Count}_{\text{clip}}(\text{n-gram}) = \text{Count of n-gram clipped by the maximum reference count}$$
$$\text{Count}(\text{n-gram}_0) = \text{Count of n-gram in the candidate translation}$$

- w_n is the weight for n-grams.

Since its emergence, BLEU has been heavily criticized for its lack of reproducibility. Some of the factors impeding reproducibility are inseparable from the metric, for instance, its dependence on the choice, quality, and number of references. However, there are many technical factors that significantly influence the metric. Firstly, there are parameters within the metric such as maximum n-gram order or the way the maximum reference count is defined; secondly, the metric heavily depends on pre-tokenization procedures or other normalization of the texts, such as case folding. All these parameters can be controlled, and in 2018, the SacreBLEU package Post [2018] was published for these purposes. It allows for a more interpretable and reproducible BLEU (and a number of other surface-based) scores by allowing to define the parameters for the metric computation and text normalization. All defined (and default) parameters are output together with the metric value as a "signature" at each run of the evaluation.

We use this package for our extrinsic evaluation, with all BLEU parameters (such as no pre-tokenization and smoothing values) equal to the default values. The only parameter that we tweak is `case`: whether the reference and hypothesis are compared intact with respect to casing or are lower-cased. We computed two versions of the BLEU metric. The one with the casing being intact (`mixed` value), which we will refer to as **BLEU**, and another with the lower-cased documents (`lc` value), which hereafter will be called **lc(BLEU)**.

chrF and Its Lower-Cased Version

One of the disadvantages of the BLEU score is its dependency on pre-tokenization, as well as on its insensitivity to the cognates or similar forms of words. Although it is almost an inseparable problem of the surface-based metrics (with the exception of the thesaurus-enhanced metrics such as METEOR), we can reduce this insensitivity by going from the word level to the character level. Of various character-based n-gram metrics for MT, one of the most popular is chrF, the character-based n-gram F score proposed in Popović [2015]. It is formulated as follows:

$$\text{chrF}_\beta = \frac{(1+\beta^2) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{2.2}$$

where:

- Precision is percentage of n-grams in the hypothesis which have a counterpart in the reference;
- Recall is percentage of character n-grams in the reference which are also present in the hypothesis;
- β is a parameter which assigns β times more importance to recall than to precision.

As we can see, this approach has a more straightforward way of handling the precision-recall balance compared to BLEU. The approach also has significantly less parameters of uncertainty; however, there are still factors such as beta and the length of the character n-grams that can be explicitly specified for a parametric evaluation.

SacreBLEU package allows for running the controlled version of chrF as well. We used this metric in two variants in our experiments by fixing all parameters to the default. The only exception is the `case` parameter. Similarly as in BLEU, we apply two variants of the metrics – with matching the casing (`mixed` value) and after lowercasing the hypothesis and the reference (`lc` value). In the first case, we refer to the metric as **chrF**, in the second case as **lc(chrF)**

COMET

The last extrinsic metric Rei et al. [2020] used in our work differs significantly from BLEU and chrF, since, firstly, it is neural and, secondly, it takes into account both the source and the reference text. The model is based on the combination of the Transformer networks: an Estimator and a Translation Ranking model. The Estimator model takes on the hypothesis, source and reference sentence, passes it through a pretrained cross-lingual encoder, creates four features (element-wise source-by-hypothesis and reference-by-hypothesis products, absolute element-wise source-hypothesis and reference-hypothesis differences), concatenates these features, and passes them to a regressor, which is trained by minimizing the mean squared error. The task of this module is to produce the absolute score for a given translation.

The Translation Ranking model is needed to enforce the distinction between the hypotheses of different qualities given the same source and reference. It takes as input four segments: source and reference "anchors" and two hypotheses, one of which is known to be better than the other. Each of the four segments is passed through the pre-trained encoder network and the pooling layer, and then the triplet margin loss introduced by Schroff et al. [2015] is applied. The triplet margin loss is the sum of two losses:

$$\mathrm{L(s,\ h^{+}}, h^{-}) = \max\left(0, d(s, h^{+}) - d(s, h^{-}) + \epsilon\right) \tag{2.3}$$

$$\mathrm{L(r,\ h^{+}}, h^{-}) = \max\left(0, d(r, h^{+}) - d(r, h^{-}) + \epsilon\right) \tag{2.4}$$

where:

- s and r are the "anchors" – source and reference sentence embeddings, respectively

- h^+ is a better and h^- is a worse translation candidate sentence embeddings
- $d(x, y)$ represents the distance between two embeddings x and y.
- ϵ is the margin parameter.

In other words, during training, the embeddings are trained in such a way that the distances between the worse translation and the “anchors” are at least by a margin bigger than the distances between the better translation and the “anchors”.

During inference, since only one hypothesis sentence is provided at each time, the harmonic mean is computed:

$$f(s, \hat{h}, r) = \frac{2 \times d(r, \hat{h}) \times d(s, \hat{h})}{d(r, \hat{h}) + d(s, \hat{h})}$$

(2.5)

where:

- $d(s, \hat{h})$ is a distance between the source and hypothesis sentence embeddings.
- $d(r, \hat{h})$ is a distance between the reference and hypothesis sentence embeddings.

In order to bound the score between 0 and 1, the following normalization is applied:

$$\hat{f}(s, \hat{h}, r) = \frac{1}{1 + f(s, \hat{h}, r)} \tag{2.6}$$

In our work, we are using the WMT22-COMET-DA model presented in Rei et al. [2022] and available at Huggingface[15]. The architecture of the basic COMET model is enhanced by the sequence tagging estimator and a QE module. The resulting system is an ensemble of two models, an ensemble between a COMET Estimator model trained on top of XLM-R Conneau et al. [2019] using WMT direct assessment data from 2017 to 2020, and a sequence tagging model trained on top of InfoXLM Chi et al. [2021]. The range of supported languages (inherited by XLM-R) comprises more than 100 languages, including Czech and Ukrainian.

The metric is entirely sentence-level; however, it also outputs the average score for the whole document. We used only the document-level COMET score in our analysis.

[15]`https://huggingface.co/Unbabel/wmt22-comet-da`

2.4.2 Intrinsic Evaluation

As was pointed out earlier, the analysis of the intrinsic metrics of subword tokenization is still an ongoing subfield, and there are numerous attempts to estimate the optimality of the tokenized sequences and subword vocabularies. In 1.3.2, we provided the discussion of the crucial aspects of internal evaluation and tokenization, namely losslessness, sequence length, and token distribution estimation. Given the text normalization described in 2.1, the algorithm construction, and the language pair of interest, our algorithms are either completely lossless or have the number of lossy transformations up to several hundred occurrences on the whole T8M dataset (and no loss on the validation and test sets). Thus, we are not providing any explicit estimation or comparison of the solutions over this parameter, assuming that all our systems are lossless[16].

Thus, in the following, we will present the metrics that are supposed to estimate the sequence length and token distribution estimation. Moreover, since we are applying various types and combinations of flags in some of our setups, there will be a specific metric dedicated only to the flag distribution for the experiments where it is applicable. The definitions of metrics and the motivation for the particular choice of them are partially inspired by the recent research of Balhar [2023] about the tokenization for the multilingual LLMs.

Characters per Token and Its Variants

This metric aims to estimate the optimality of the encoded text in terms of its length. The concept of the metric is to evaluate the number of characters in the initial text that an average token comprises. The mathematical formulation of the metric is taken from Balhar [2023][p. 27] and is shown below:

$$CPT(\tau, \pi, C) = \frac{\sum_{s \in C} |s|}{\sum_{s \in C} |\tau(\pi(s))|} \tag{2.7}$$

where:

- τ is a given tokenizer,
- π is the preprocessing function such as InCa or InDia (where applicable, for no preprocessing scenario $\pi(s) = s$),
- C is a given language corpus,
- s is a sentence within the C corpus, $|s|$ is its length in characters and $|\tau(\pi(s))|$ is the length of the encoded sequence in tokens.

Since we expect a better tokenizer to minimize the space of the encoded sequence, we say that a better tokenizer should have a higher number of characters per token ratio. The metric has a lower bound of 1 (the worst case where each character corresponds to its own token).

Balhar [2023] shows in his analysis that two other frequently used metrics, average sequence length (i.e. average number of tokens per sentence) and the

[16]However, for the future research in a more multilingual setting, we should bear this objective in mind again and possibly create another metric for this aspect of pre-processing and tokenization.

word fertility (average number of tokens per word) can be mathematically reduced to the CPT formula. The two advantages that CPT has is that, firstly, it is independent from the notion of "word" which is language-specific and problematic for many non-European languages; secondly, it is easier to interpret due to its clear lower bound (contrary to the average sequence length).

The metric is applied to tokenized validation and test sets such as `flo`. In other words, how long are the tokens that are used at the inference step for a given file. However, we may also be interested in looking at how long on average the subword vocabulary items that the tokenizer chose as a result of training; that is, what is the average length of the token that the tokenizer would expect at the inference step. Notably, since we are not applying CPT to the training dataset as we compare the intrinsic and extrinsic metrics on the smaller datasets, this ratio can give us an indirect estimation on the average plausible length of the token on the training dataset. Thus, we introduce a metric that, contrary to the one described above, we will apply to the tokenizer vocabulary, not to the encoded texts:

$$CPT_v(V) = \frac{\sum_{t \in V} |t|}{|V|} \tag{2.8}$$

where:

- V is a subword vocabulary trained by the tokenizer τ, which is trained on the (usually preprocessed) training dataset $\pi(C_{training})$, and $|V|$ is the length of the vocabulary in subwords,
- t is a unique subword in the tokenizer vocabulary, and $|t|$ is its length in characters.

Thus, we will be using the CPT metric as a default metric for our intrinsic analysis of tokenized texts, and the CPT_v metric for analysis of the vocabularies that are trained on our datasets.

Average Rank

The next metric is, in our classification, related to the token distribution estimation. We are interested in understanding how uniform our distribution is, as we hypothesize that the more uniform the empirical distribution of tokens, the more informative each of them (contrary to the distributions that are closer to a skewed one, where overly frequent tokens would not be informative and overly rare tokens would be met too sparsely distributed over the text). We can estimate this distribution by ranking the empirical token distribution in a given text. One of the easiest metrics proposed for estimating the share of effectively used tokens is counting the least frequency in the 95th percentile of the most frequent classes, following Gowda and May [2020]. However, for this metric to be informative, we need a reasonably large dataset to estimate, as the texts comprising hundreds or even thousands of sentences will most likely not differentiate the tokenizers as the count will always be low. This was the case of our dataset; thus we dismissed this metric.

Another simple metric that we can take from the ranked distribution of the tokens in the text is the average rank of the text. It is formally represented in the formula below:

$$AR(\tau, C) = \sum_{t \in V_\tau} rank(t, \tau(C)) \cdot p(t, C) \tag{2.9}$$

where

- τ is a tokenizer function, and V_τ is its vocabulary,
- C is a given corpus,
- $rank(t, \tau(C))$ is a rank of a token t (position in the list of the unique tokens met in tokenized corpus C ordered by frequency),
- $p(t, C)$ is a frequency of a given token in the corpus.

In other words, the average rank metric is a weighted average of the tokens met in the tokenized corpus, where weights are the frequencies of the tokens in the given corpus. If the distribution is skewed, then it will have a long tail of tokens with small probabilities; in this case the bigger frequency weights will be skewed towards the head of the distribution. The more uniform the distribution (or at least the smaller the tail in favor of the high-frequency tokens), the larger the weighted average. Thus, we expect that the higher average rank of the tokenized text would signify the more optimal usage of the tokens, hence a better tokenizer.

Rényi Efficiency

The metrics such as frequency of the 95th percentile token or average rank, despite being transparent and easy to calculate, may not be informative about the whole token distribution. Recently, a rigorous analysis of tokenization from the perspective of information theory was performed Zouhar et al. [2023], in order to create an intrinsic metric that would comprise more information on the tokenized text. The authors start with the assumption that, contrary to many other NLP tasks which are usually modeled in terms of the noisy channel, most up-to-date tokenization algorithms perform the noiseless transformations (i.e., the full round of tokenization and de-tokenization returns the initial text). Thus, contrary to the optimization task in the noisy channel where we need to minimize the information loss, in the noiseless case, we only care about the efficiency of the channel. For this efficiency, two parameters are important, the frequency distribution of the tokens and the length of each token used in the tokenization function. The efficiency is formulated in terms of entropy and the optimal efficiency is proven to be bounded by the entropy of the uniform distribution of the tokens with the same code length. The authors show that the Shannon entropy is not an optimal entropy model for that task, as it linearly penalizes the long codes, which may not be an optimal thing in the case of the infrequent tokens. Their solution is to use the generalized version of the Shennon entropy, namely, Rényi entropy, which allows us to give flexibility to the preferred expected token length (as we may expect that for different tasks, different tokens may be preferred).

The final metric is defined as follows:

$$\mathrm{EFF}_\alpha(p_{\Sigma^*}, t) = \frac{L_{enc^s_\Delta}(p_{\Delta^*})}{L_{enc^U_\Delta}(p_{\Delta^*})} \tag{2.10}$$

where

- $t : \Sigma^* \to \Delta^*$ is a tokenization function, which maps texts in alphabet Σ to sequences of tokens in Δ,
- p_{Σ^*} is a distribution of the input sequence Σ^*,
- (p_{Δ^*}) is the frequency distribution of the tokens given tokeniser function t,
- L_{enc_Δ} is an expected code length:

$$L_{\mathrm{enc}_\Delta}(p_\Delta) = \sum_{\delta \in \Delta} p_\Delta(\delta) \cdot |\mathrm{enc}_\Delta(\delta)|$$

- $L_{\mathrm{enc}^U_\Delta}$ is an expected code length for a uniform encoded (which assigns the same lengths for all tokens),
- $s = \alpha^{-1} - 1$, where α is an order of the Rényi entropy,
- $L_{\mathrm{enc}^s_\Delta}$ is a discounted expected code length for a given s, which allows for non-linear penalization of the longest tokens.

Since the most informative distribution is the one with the tokens of uniform lengths (the one in the denominator), it is the upper bound of the observed distribution. Thus, the scope of the metric is between 0 and 1, and the closer the score to 1, the more optimal it is believed to be.

The empirical analysis conducted by the authors showed that for the MT task, the preferred value of alpha equals 2.5; in this case, the Pearson correlation between Rényi efficiency and BLEU score equals to 0.78.

The authors provided the community with a standalone Python package that allows us to compute the Rényi efficiency score[17]. We used this package for the computations and we used the default alpha parameter (2.5) to estimate the efficiency of the distribution.

Flag-Related Statistics

Since both our suggested approaches (InCa and InDia) and the competing approaches such as TokenMonster capcodes or Marian inline casing use flags of certain types at the preprocessing step, we intend to pay special attention to their positions in the tokenized texts. However, the principles of the flags' assignment and their internal structure differs significantly (only two codes for TokenMonster, up to four codes in InCa and four codes with different semantics in Marian inline casing, and as for InDia, the flag can consist of the whole sequence of the symbols), we did not intend to create a formalized metric that would compare the distributions of the flags in different systems. Thus, where applicable, we focused on mostly looking at the number of the unique tokens comprising the flags used

[17]`https://github.com/zouharvi/tokenization-scorer`

ID	**Level**	**Short Description**	**Range**
BLEU	ext	BLEU score	0-100
lc(BLEU)	ext	lower-cased BLEU	0-100
chrF	ext	chrF score	0-100
lc(chrF)	ext	lower-cased chrF	0-100
COMET	ext	COMET score	0-1
CPT	int	characters per token	1-∞
CPT$_v$	int	average token length in vocabulary	1-∞
AR	int	average rank	1-$\|V\|$
EFF	int	Renyi efficiency	0-1
R(f)	int	rank of a flag f	1-$\|V\|$
C(F), C$_V(F)$	int	count of unique flags in preprocessed text, vocabulary	1-$\|V\|$

Table 2.4: Overview of the metrics used in the analysis. The names in the "ID" column will be consistently used to denote the metrics hereinafter. The "Level" column shows whether the metrics perform the extrinsic (`ext`) or intrinsic (`int`) evaluation. The column "Range" shows the range of the possible values of the metrics ($|V|$ stands for the tokenizer vocabulary size). For all metrics except for flag ranks the higher value means the better performance.

in the tokenized texts, as well as the ranks of these tokens. For uniform notation, when the discussion is about the ranks of a particular flag, it would be denoted as $R(flag)$. In case of the counts of all unique flags, either in the observed distribution of the tokenized text, or in the tokenizer vocabulary, it would be denoted either as $C(F)$ for the tokenized text and $C_V(F)$ for the vocabulary entries.

2.5 Implementation Details

We use the Marian implementation Junczys-Dowmunt et al. [2018] of the Transformer model Vaswani et al. [2017]. Marian is a fast (due to its realization in C++) implementation of the main NMT architectures, including Transformer, and it provides the end-to-end training, validation, and inference system for NMT. We are using the Marian version v1.12.14 which was released in November 2023. Here are the main hyperparameters of our model's implementation:

- 9 GB of the preallocated GPU memory;
- transformer-base model size;
- tied embeddings and output layer;
- validation – every 5000 updates;
- validation metrics: cross-entropy and the above-mentioned sacreBLEU implementation of BLEU score;
- mini-batch size (for both training and decoding) equals to 40 sentences;
- beam search with beam size 4 applied (for both training and decoding);

- attention scores are normalized;
- maximum length of a sentence at the training step – 150 tokens, in a validating sentence pair – 500 tokens;
- pre-loading 1000 mini-batches to sort lengths within each mini-batch is enabled;
- training lasts 16 epochs (for data augmentation – 4 epochs X quadrupled data), where each epoch is defined as single full pass of the training dataset. The only exception is the data augmentation scenario, where we use the training dataset of the size of four initial datasets; in this case the training lasts 4 epochs (yielding the same 16 full passes of the initial dataset size).
- no early stopping is enabled.

All other parameters are set to default, described in the Marian documentation.[18] All models were trained with a fixed seed. We could not fix the size of the tokens in the mini-batch due to the conflicting hyper-parameters and the implementation details of Marian; thus the number of the tokens in a mini-batch was defined for each system separately and on average was between 7500 and 8500 tokens.

Marian system allows for both using an in-built SentencePiece tokenizer instance or a text which is tokenized by an external system. In most cases, we were using the in-built SentencePiece tokenizer, which is Marian's for of the SentencePiece,[19] version 0.1.94. The only exception tackled TokenMonster, as it generates tokenized text itself. SentencePiece tokenizer parameters are described in 2.3, and the training lasted 40 to 60 minutes.

All tokenization and MT experiments were run on a GPU cluster with a single GPU used for one experiment. The NVIDIA RTX 3090 was used for all experiments, with driver version 525.85.12 and CUDA Version: 12.0. The training time typically spanned 22 to 25 hours.

[18] `https://marian-nmt.github.io/docs/cmd/marian/`

[19] Available at: `https://github.com/marian-nmt/sentencepiece`

3. Issues in Stabilization and Normalization

3.1 Data Normalization: Apostrophes and Inconsistent Casing

The method of the data normalization that was used for the creation of the training data (Popel et al. [2022]) did not include several features that are important for our research. One of these features is specific to Ukrainian language, while another tackles the general problem of Latin- or Cyrillic-based languages.

3.1.1 Inconsistencies with Apostrophes in Ukrainian data

The first problem covers treating apostrophe symbol in Ukrainian. The Ukrainian orthography includes an apostrophe sign which is mandatory in writing. It usually has phonetic meaning in context with iotated vowel characters such as я or ї, negating the palatalization of preceding vowel and releasing the [j] sound (compare: a syllable бя is pronounced as [b^ja] without apostrophe, whereas б'я is pronounced as [bja]).

The character was included in one of the first versions of Unicode (version 1.1, June 1993) under the name of “MODIFIER LETTER APOSTROPHE” and the code U+02BC, and was assigned to the category of modifier letters. However, the user experience of many Ukrainian speakers changed the seemingly straightforward situation for character usage. For instance, the Ukrainian language support in the IT products was always significantly weaker compared to Russian; thus many people (especially in the first decades of the computer era) had to use the Russian+English keyboards to type Ukrainian texts. In this setup, a logical solution was to use an English apostrophe ' U+0027 (which we will later refer to as ASCII apostrophe since it was already present there) or quotation marks from the English or Russian keyboards. A human reader would not distinguish from a modifier letter apostrophe; however, a computer would see them as different characters. Another factor is inconsistency of the location of apostrophes (both the “ASCII” and the “Ukrainian” ones) on various keyboards provided by various operating systems and especially by various smart phones (IPhone, Android- or Windows-based systems). In such cases, a logical user experience is to find the fastest way to type an apostrophe, no matter what UTF code it has, which sometimes ends with using English apostrophes or quotation marks. Because of these reasons, to name a few, the Ukrainian language data in the Internet comprise large inconsistencies of character usage with respect to the apostrophe.

What is especially important for the topic of our research, most of the alternatives for a modifier letter apostrophe have another category of the Unicode characters, mostly – punctuation marks. This is a crucial complication for our work, since on all levels of MT process (our design of preprocessing algorithms, subword tokenizers and extrinsic evaluation systems), most of the systems have in-built distinction between the alphanumeric and non-alphanumeric characters, with the latter often use as the token boundary. Thus, the same Ukrainian word written with the modifier letter apostrophe and the ASCII apostrophe, would not only consist of various characters, but in most cases would be even treated as two or three tokens instead of one. Since tokenization is our main focus, we understood that it is necessary to normalize the training data with respect to apostrophes.

Speaking of the apostrophe normalization, we had to decide which character we should map the inconsistent apostrophes to. The practice of the most of the Ukrainian language data resources, such as Aranea web-corpora or GRAK (General Regionally Annotated Corpus of Ukrainian), shows that the corpora creators tend to normalize all apostrophe variants to the ASCII apostrophe. The paper Starko et al. [2021], which describes the text normalization schema in GRAK (mainly concentrating on OCR problems), also specifies this solution and presents a package that covers a number of other frequent normalization problems for Ukrainian, such as letter "i" handling.[1] Finally, the test sets used in the Czech-Ukrainian WMT competitions also contain consistent use of ASCII apostrophes or predominantly use this character. We could not see any explicit justification for mapping to ASCII apostrophe specifically; however, we can hypothesize that it can be done based on, firstly, preliminary analysis on the frequency distribution of the apostrophe variants in the datasets; secondly, the user stories of the language data resources by the international users who may have only English keyboard. We checked the counts of the most widely mentioned substitutes of the modifier letter apostrophe and the letter apostrophe itself, in our main training dataset (T8M) and in the default validation dataset (flo) by checking the Ukrainian-only data and only the allowed contexts for the apostrophe. The results are shown in Table 3.1. We can see that the ASCII apostrophe is the most frequent version, and the modifier letter apostrophe comprises less than 0.1 percent of the whole number of occurrences.

Based on the best practices and the observations on our data, we, first, decided to normalize the test dataset by unifying all occurrences to the ASCII apostrophe. As for the training data, in our main setup we decided to normalize the data to an ASCII apostrophe as well. Our goals were twofold: firstly, we wanted to have a more interpretable and searchable dataset in terms of the apostrophe usage; secondly, we hoped to improve the extrinsic and intrinsic metrics by such a normalization. To assess the achievement of the latter, we compared the non-normalized data to the ASCII normalization of the apostrophes. For each of the scenarios, we compare the default MT pipeline (without any pre-processing) and the standard InCa pre-processing. The results are shown in Tables 3.2 and 3.3

[1]Another problem which roots in similar factors of the keyboard usage as described above, tackles the letter "i", which is present in Ukrainian alphabet but absent in the Russian alphabet. For this reason, a Cyrillic letter "i" was introduced to the UTF charset under the names "CYRILLIC (SMALL/CAPITAL) LETTER BYELORUSSIAN-UKRAINIAN I" and IDs U+0456 and U+0406. These letters differ from the Latin letters "i" and "I" under IDs U+0069 and U+0049, respectively. However, if a person has only a Russian and an English keyboard, a logical user story is to type a Latin letter "i" instead of the Cyrillic one. We found such occurrences of inconsistencies in our training data; however, we decided not to normalize them for a number of reasons. Firstly, the estimated number of Latin "i" occurrences is orders of magnitude lower than the correct ones (for instance, for small letters it is 17,000 Latin letters against over 25,000,000 Cyrillic ones, which should not drastically drop the performance of the algorithm but may even add robustness. Secondly, the rules for the correct detection and substitution of the letter "i" are more complicated than the rules for apostrophes.

UTF code	Name	Count (T8M)	Count (flo)
U+0027	APOSTROPHE	480228	124
U+2019	RIGHT SINGLE QUOTATION MARK	108649	3
U+2018	LEFT SINGLE QUOTATION MARK	501	0
U+02BC	MODIFIER LETTER APOSTROPHE	493	0
U+2032	PRIME	54	0
U+0384	GREEK TONOS	6	0

Table 3.1: Table 3.1. Counts of occurrences of various types of apostrophes in the Ukrainian parts of the training (T8M) and test (flo) datasets

for Czech-to-Ukrainian and Ukrainian-to-Czech translation directions, where we compare the main intrinsic and extrinsic metrics in the default flo datasets. The values "-" and "ASCII" relate to the absence of normalization and normalization to ASCII apostrophes, respectively.

We can see as a result of the comparison that normalization of apostrophes did not bring any significant improvement in any of the intrinsic or extrinsic metrics. However, since it did not decrease either and since we now have more understanding of what the expected functionality of the apostrophe is, we decided to keep the normalized version of the training data hereinafter for the experiments.

We also wanted to clarify whether mapping the apostrophes to the "orthographically correct" Ukrainian modifier letter apostrophe would improve the performance of the algorithm. The motivation behind that was to estimate the possible benefits of using the alphanumeric category of characters (which Ukrainian letter apostrophe is part of) instead of punctuation (which ASCII apostrophe is part of). To do that, we substituted all occurrences of the ASCII apostrophe in the apostrophe context with the Ukrainian apostrophe, hoping that this would automatically increase the number of subword tokens with those containing a modifier letter. However, we discovered that SentencePiece does not differentiate between the alphanumeric and non-alphanumeric words (which would make difference for ASCII and Ukrainian apostrophe). Instead, it only differentiates between the Unicode scripts (which is the function of split_by_unicode_script parameter), and the Modifier letter apostrophe, although being in the alphanumeric character type, is not in the Cyrillic script.

Thus, to allow for generalization of the modifier letter apostrophe in the subwords, we first had to deactivate the split_by_unicode_script parameter in SentencePiece. Since this change of the parameter may help by itself, we made two next ablations on our experiment: firstly, by changing the split_by_unicode_script parameter to false (while preserving the ASCII apostrophe), and secondly, by changing the split_by_unicode_script parameter together with substituting the apostrophe to the Ukrainian modifier letter. As a result, the crucial parameter was the value of the split_by_unicode_script parameter, and using the ASCII or Ukrainian apostrophe did not change any of the statistics. Thus, we provide only one of these two experiments (with turning the split_by_unicode_script parameter off and keeping the ASCII apostrophe) and present it in the same

inca	normalization	CPT	AR	EFF	BLEU	chrF	COMET
-	-	3.974	1236	0.538	21.7	51.5	0.872
-	ASCII	3.973	1238	0.538	21.6	51.3	0.869
-	*ASCII, -split*	4.065	1318	0.577	21.6	51.5	0.872
+	-	4.009	1172	0.526	21.6	51.3	0.870
+	ASCII	3.995	1166	0.522	21.7	51.4	0.870
+	*ASCII, -split*	4.092	1256	0.549	21.5	51.2	0.869

Table 3.2: Internal and external metrics comparison with respect to apostrophe normalization, Czech-Ukrainian translation direction. The first two columns show the training parameters of the MT model, each line corresponds to a separately trained system. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Normalization" shows whether we used no mapping to the only apostrophe (-), mapping to the ASCII apostrophe (ASCII), mapping to ASCII apostrophe with disabled SentencePiece parameter "split_by_unicode_script" (ASCII, -split). The metric naming is defined in 2.4.

inca	normalization	CPT	AR	EFF	BLEU	chrF	COMET
-	-	4.033	1189	0.516	22.8	51.2	0.872
-	ASCII	4.033	1189	0.516	22.7	51.0	0.873
-	*ASCII, -split*	4.189	1292	0.572	23.0	51.1	0.873
+	-	4.034	1121	0.503	22.6	50.7	0.867
+	ASCII	4.014	1113	0.500	22.7	51.0	0.867
+	*ASCII, -split*	4.166	1217	0.534	22.7	51.0	0.870

Table 3.3: Internal and external metrics comparison with respect to apostrophe normalization, Ukrainian-Czech translation direction. The notation is equivalent to the one in Table 3.2

tables 3.2 and 3.3 as the italicized lines.

What we can see as the result of these experiments is that while the extrinsic metrics remain constant, all intrinsic metrics show increasing performance compared to the activated split_by_unicode_script parameter. On the other hand, the difference between using the ASCII and Ukrainian apostrophe given the negative split_by_unicode_script parameter does not show a significant difference. We can see the explanation for this if we examine the SentencePiece dictionaries formed by these setups in the table 3.4. The table shows that the number of tokens with an apostrophe disregarding the apostrophe symbol amounts to more than 170 tokens, which is approximately 0.5% of the whole 32,000 token subword vocabulary. The comparison of the tokens with apostrophe mapped to ASCII and to the Ukrainian letter shows that they overlap significantly (100 to 110 of tokens out of approximately 170, depending on setup), which demonstrates us that choice of the ASCII or Ukrainian apostrophe does not play a significant role for SentencePiece given a disabled function of splitting by Unicode script[2]. However, turning this parameter off comes at a cost of a slight

[2]The difference in the tokens can be explained by the fact that not all occurrences of ASCII and Ukrainian apostrophes overlap, since there are occurrences of the apostrophised foreign words in the Ukrainian texts, and there are even more cases of the apostrophe usage in the Czech data which have nothing to do with the Ukraininan apostrophe.

decrease in the average token length in the vocabulary (by 0.02 characters in the default setup and by 0.04 in the InCa setup). This happens because since Unicode script division is disabled, all frequent combinations of any other punctuation with alphabetic symbols are now also taken into account. We can see evidence of that in the vocabularies: if we take the non-preprocessed vocabulary trained on ASCII-normalized apostrophe and compare the two versions with and without split_by_unicode_script disabling, we will see that the difference between the unique tokens comprises 3284 tokens (10% of the vocabulary). Of these 3284 tokens, in the disabled split_by_unicode_script scenario, almost 1000 comprise shorter (2- or 3-character) sequences that combine a punctuation mark and a character (for instance, “м.” or “_„N”. Only after filling vocabulary with them, it proceeds with longer alphanumeric-only subwords. The vocabulary trained with split_by_unicode_script enabled does not have such tokens, thus offering more space to the longer alphanumeric sequences instead of the “punctuation+character” combinations. With that, the average token length of the tokens unique for the vocabulary trained with enabled script splitting is 6.56, while the unique tokens trained with disabled script splitting are 6.37 characters on average.

The takeaways of this comparison are contradictory. On the one hand, enabling the SentencePiece tokenizer to include the apostrophe allows one to include frequent tokens with it to the vocabulary and thus to catch longer Ukrainian subwords as a single unit. It may be especially precious, as most of the extracted subwords containing an apostrophe have it within the stem (for instance, subwords like “об’єкт” (object), “п’ять” (five), “пам’ять” (memory) and their cognates), not on the morphemic borders. However, this comes at the cost of spending several times more tokens on the frequent but less plausible combinations of single alphabetic characters and punctuation, which leads to decreasing the average token length of the vocabulary. Thus, since there was no conclusive advantage of switching to the disabled splitting, we stuck to the ASCII normalization and default (enabled) splitting by Unicode scripts in the SentencePiece tokenizers in all experiments hereafter.

Still, the general consideration on this issue is that, despite some tokenisers such as SentencePiece provide a wide range of tuning parameters for the tokenization depending on various information from UTF (such as normalization or script types), language-specific “artifacts” that do not fall into one of the preprogrammed categories may appear in a particular NLP case, which would require targeted preprocessing of the data or tuning the tokenizer algorithms. Notably, such work cannot be tokenizer-agnostic (which we hoped to perform by substituting the ASCII apostrophe to the Modifier letter apostrophe). Probably, the solution that should take all advantages and omit all disadvantages related to the apostrophes in Ukrainian, for our setup, is substituting the all Ukrainian apostrophe occurrences with a specific Cyrillic letter absent in the Ukrainian but present in the “Cyrl” script in Unicode, and enabling the split_by_unicode_script parameters. However, since this does not seem to be a crucial issue in the intrinsic or extrinsic metrics, we did not proceed with such experiments in this work.

InCa	Apostrophe Normalization	Apostrophe	CPT_v	Tokens with Apostrophe
-	-	ASCII	6.837	2
-	ASCII	ASCII	6.837	2
-	ASCII, -split	ASCII	6.818	176
-	letter, -split	letter	6.818	172
+	-	ASCII	7.124	1
+	ASCII	ASCII	7.119	1
+	ASCII, -split	ASCII	7.081	178
+	letter, -split	letter	7.081	171

Table 3.4: SentencePiece vocabulary statistics on apostrophe normalization modes. The first two columns show the training parameters of the SentencePiece model, each line corresponds to a separately trained tokenizer. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Normalization" shows whether we used no mapping to the only apostrophe (-), mapping to the ASCII apostrophe (ASCII), mapping to ASCII apostrophe with disabled SentencePiece parameter "split_by_unicode_script" (ASCII, -split) or mapping to the Ukrainian modifier letter apostrophe with the disabled "split_by_unicode_script" parameter (letter, -split). The "Apostrophe" column denotes which apostrophe character we would count in the "Tokens with Apostrophe" column – ASCII one (ASCII) or the "Modifier letter apostrophe" (letter). "CPT_v" denotes the average token length in the vocabulary, "Tokens with Apostrophe" show the number of the tokens that contain the apostrophe of a given type.

3.1.2 Inconsistencies with Fully Upper-Cased Sentences

One of the main objectives of our work is to increase the robustness of MT systems against various casings. According to the preliminary analysis on our training data, we observed that, out of the three main scenarios of casing noise, the hardest task for the MT systems was to translate the fully upper-cased texts to their fully upper-cased counterparts.

We hypothesized that this could happen because of the lack of fully upper-cased sentence pairs in the training data. The initial dataset contained sentence pairs that were either both fully upper-cased or only upper-cased on one of the sides. The total of such sentence pairs of all types is equal to approximately 17,000. Only 7,900 of these sentence pairs were upper-cased consistently (both in source and target). We decided to normalize the inconsistencies of this type and make sure that the upper-cased sentences have their upper-cased counterparts.

The results of this operation are shown in Tables 3.5 and 3.6 for the Czech-Ukrainian and Ukrainian-Czech translation directions. We compared the changes of the extrinsic metrics on the fully upper-cased flo dataset for two scenarios: no pre-processing and standard InCa preprocessing. For reference, we also provide information on the scores of the same metrics on the default (non-upper-cased) flo dataset to see if the quality on the basic dataset decreases.

We can see that in both setups, for all translation directions, the performance on the BLEU, chrF and COMET scores increases significantly. If we take a look at the no-pre-processing setup, we see that chrF and COMET figures as they gain from 4 to 7 and from 0.02 to 0.03 points, respectively. However, even larger increase occurs with InCa preprocessing, as the versions of BLEU, chrF and COMET metrics reach the levels of performance on the default (non-upper-cased) dataset. This shows us that even a relatively small but consistent number of paired upper-cased sentences can help increase the performance of the MT systems, especially if they are enhanced by the preprocessing techniques. Based on that, we decided to take on such an upper-case normalization and used it for all experiments hereinafter.

3.2 Stabilization of the Experiments

The initial results of our normalization experiments, as well as the comparison of the non-preprocessed and standard InCa models on the default flo dataset, showed us that the performance on the extrinsic metrics remained within a small interval and did not show significant improvement. Moreover, the first experiment in InCa showed a slight decrease compared to the no-preprocessing setup. We decided to estimate how stable and reproducible the MT systems are given the fixed training setup, and whether we should rely on the slight changes of our extrinsic metrics.

To do that, we ran the non-preprocessing and InCa experiments with default parameters, trained on our dataset (after applying all normalization described above), three times each. After that, we conducted an extrinsic evaluation of the systems both with our general metrics and with the pairwise bootstrap re-sampling technique, which was presented in Koehn [2004] and is implemented in

InCa	Norm. Upper	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
-	-	$1.3_{22.1}$	$2.4_{22.7}$	$18.8_{51.7}$	$23.4_{52.2}$	$0.421_{0.874}$
-	+	$1.6_{21.6}$	$1.9_{22.1}$	$22.5_{51.3}$	$23.0_{51.8}$	$0.448_{0.869}$
+	-	$2.5_{21.4}$	$21.4_{22.1}$	$6.2_{51.2}$	$51.2_{51.9}$	$0.578_{0.868}$
+	+	$21.3_{21.7}$	$21.3_{22.4}$	$51.3_{51.4}$	$51.3_{52.1}$	$0.871_{0.87}$

Table 3.5: External metrics comparison with respect to upper-case paired sentence normalization, Czech-Ukrainian translation direction. The first two columns show the training parameters of the MT model, each line corresponds to a separately trained system. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Norm. Upper" shows whether we used upper-case preprocessing for the consistency of source and target sentence pairs (- for non-normalized data, + for normalized data). The models are evaluated on the fully upper-cased flo dataset and on the default flo dataset (described in Section 2.1); since the main interest is fully upper-cased dataset, it is provided in general font, while the subscripts show the performance on the default dataset for reference. The metrics are decscribed in Section 2.4.1.

InCa	Norm. Upper	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
-	-	$1.2_{23.1}$	$3.4_{23.6}$	$14.9_{51.4}$	$25.0_{51.9}$	$0.389_{0.876}$
-	+	$1.9_{22.7}$	$2.5_{23.2}$	$21.8_{51.0}$	$23.2_{51.5}$	$0.419_{0.873}$
+	-	$1.9_{23.0}$	$22.9_{23.6}$	$4.5_{51.1}$	$51.4_{51.8}$	$0.579_{0.866}$
+	+	$22.8_{22.7}$	$22.8_{23.3}$	$51.3_{51.0}$	$51.3_{51.7}$	$0.865_{0.867}$

Table 3.6: External metrics comparison with respect to upper-case paired sentence normalization, Ukrainian-Czech translation direction. The notation is equivalent to the one described in Table 3.5.

	BLEU			chrF			COMET
Run	Score	$\mu \pm 95\% CI$	p-value	Score	$\mu \pm 95\% CI$	p-value	Score
1st	21.5	21.5 ± 1.1	-	51.2	51.2 ± 0.7	-	0.871
2nd	21.8	21.8 ± 1.1	0.076	51.5	51.5 ± 0.7	0.023*	0.873
3rd	21.5	21.5 ± 1.0	0.274	51.3	51.3 ± 0.8	0.271	0.873

Table 3.7: Extrinsic metrics comparison of the MT systems trained in the same setup: no preprocessing, Czech-Ukrainian direction. Each line represents an instance of the model trained separately but with the same setup and hyper-parameters. The BLEU and chrF columns comprise three parameters: "Score" is an overall score computed for the whole document; "$\mu \pm 95\% CI$" is mean score and 95% confidence interval based on bootstrap resampling of the given system output; "p-value" is significance test comparing the first run with the given run (significance threshold is 0.05, statistically significant values are marked with an asterisk). The COMET value shows the score only.

	BLEU			chrF			COMET
Run	Score	$\mu \pm 95\% CI$	p-value	Score	$\mu \pm 95\% CI$	p-value	Score
1st	21.7	21.7 ± 1.0	-	51.3	51.3 ± 0.7	-	0.871
2nd	21.7	21.7 ± 1.0	0.331	51.4	51.4 ± 0.7	0.174	0.870
3rd	21.6	21.6 ± 1.1	0.298	51.2	51.2 ± 0.7	0.287	0.869

Table 3.8: Extrinsic metrics comparison of the MT systems trained in the same setup: standard InCa preprocessing, Czech-Ukrainian direction. The notation is equivalent to one in 3.7.

SacreBLEU [3] package – Post [2018]. Within the bootstrap resampling approach, for a particular text, 1000 samples of translated sentences are taken and compared against the reference. For each sample, a BLEU (or chrF) score is computed; then the scores are ordered, 2.5% of the highest and lowest scores are dismissed, and the mean value of the rest of the scores is calculated. With this, we obtain an expected mean and 95% confidence interval for a given system. Pairwise bootstrap resampling is based on the same technique, but on top of that, the samples drawn from two systems are compared and for each sample pair. Our null hypothesis is that the samples from both system outputs are generated by the same process. Based on pairwise comparisons, we perform a p-value significance test with a threshold of 0.05. If the p-value shows significance, we can assume that the performance of the two systems is significantly different. This approach, contrary to a "bare" BLEU score, is believed to be more informative in terms of the reliability of the MT scores.

We compare the performance of the systems of the same setup with absolute BLEU, chrF and COMET scores, as well as with paired bootstrap-resampled BLEU and chrF scores. For each setup, each system is compared with pairwise bootstrap resampling against the first run. The results of these comparisons are shown in Tables 3.7–3.10.

The results within each of the setups show that the variation in the absolute BLEU and chrF values between the runs within the same setup may not be

[3]`https://github.com/mjpost/sacrebleu`

	BLEU			chrF			COMET
Run	Score	$\mu \pm 95\%CI$	p-value	Score	$\mu \pm 95\%CI$	p-value	Score
1st	22.6	22.6 ± 1.1	-	50.8	50.8 ± 0.8	-	0.872
2nd	22.9	22.9 ± 1.0	0.139	51.0	51.0 ± 0.8	0.134	0.871
3rd	22.7	22.7 ± 1.1	0.258	51.0	51.0 ± 0.8	0.121	0.873

Table 3.9: Extrinsic metrics comparison of the MT systems trained in the same setup: no preprocessing, Ukrainian-Czech direction. The notation is equivalent to one in 3.7.

	BLEU			chrF			COMET
Run	Score	$\mu \pm 95\%CI$	p-value	Score	$\mu \pm 95\%CI$	p-value	Score
1st	22.9	22.9 ± 1.1	-	51.1	51.0 ± 0.8	-	0.867
2nd	22.7	22.7 ± 1.1	0.207	51.0	51.0 ± 0.8	0.27	0.867
3rd	22.6	22.6 ± 1.1	0.11	50.8	50.8 ± 0.8	0.081	0.867

Table 3.10: Extrinsic metrics comparison of the MT systems trained in the same setup: standard InCa preprocessing, Ukrainian-Czech direction. The notation is equivalent to one in 3.7.

large; however, the confidence intervals show a wide variation, usually equal to 1 BLEU and 0.75 chrF score. Moreover, although the statistical significance in the paired bootstrap resampling was reached only once (second run of no-preprocessing setup for Czech-Ukrainian direction, chrF), the values which are similar to the significance level are seen in several other cases such as BLEU in the same experiment, or the third run of InCa preprocessing for Ukrainian-Czech direction. The variation range of the experiments shown in the tables covers also the normalization experiment statistics that we showed above, in Sections 3.1.1-3.1.2.

What is crucial for our research is that we can see that the absolute values and the confidence intervals of the no-preprocessing and InCa scenarios also overlap considerably. To verify this, we also merged the three runs of each of the above setups and compared such merged triplets of files between the default and InCa preprocessing methods. The results for each translation direction are shown in Tables 3.11 and 3.12. We can observe that combining the outputs of the systems allows us to narrow the variation to 0.5 BLEU points, however, the mean values for the no-preprocessing and InCa preprocessing setups remain the same (which is supported by the p-values).

On the one hand, this shows us that convincingly beating the baseline with our preprocessing technique will be a challenging task, which may sound disappointing. On the other hand, we should remember that the aims of our research were also to improve the quality on various types of noising, as well as to optimize the intrinsic parameters of tokenization. The results of the analysis on normalization of the upper-case translation and apostrophe normalization show that the improvements on InCa amount to dozens of BLEU points and substantial changes in the intrinsic metrics. Thus, hereinafter our main interest would be formulated as follows: Our main goals are to improve the extrinsic metrics on translation of the noised texts, and intrinsic metrics (in both general and noised texts), controlling that there is no significant drop in the extrinsic metrics of the

	BLEU			chrF		
Prepro-cessing	Score	$\mu \pm 95\%CI$	p-value	Score	$\mu \pm 95\%CI$	p-value
base	21.6	21.6 ± 0.6	-	51.3	51.4 ± 0.5	-
inca	21.7	21.7 ± 0.6	0.294	51.3	51.3 ± 0.5	0.197

Table 3.11: Extrinsic metrics comparison of the combined outputs of 3 runs of MT systems trained in the same setup, Czech-Ukrainian direction. The "preprocessing" column shows whether we used InCa ("inca") or no pre-processing ("base"). The BLEU and chrF columns comprise three parameters: "Score" is an overall score computed for the whole document; "$\mu \pm 95\%CI$" is mean score and 95% confidence interval based on bootstrap resampling of the given system output; "p-value" is significance test comparing the first and the second lines (significance threshold is 0.05).

	BLEU			chrF		
Prepro-cessing	Score	$\mu \pm 95\%CI$	p-value	Score	$\mu \pm 95\%CI$	p-value
base	22.7	22.7 ± 0.6	-	50.9	50.9 ± 0.4	-
inca	22.7	22.7 ± 0.6	0.417	51.0	51.0 ± 0.4	0.27

Table 3.12: Extrinsic metrics comparison of the combined outputs of 3 runs of MT systems trained in the same setup, Ukrainian-Czech direction. The notation is equivalent to one in 3.11.

general translation of the validation sets.

4. Experiments with Casing Strategies

This chapter will address experiments with the impact of various inline casing algorithms. Firstly, we will compare the main algorithm we propose, InCa, with the base MT system and the main competing inline casing solutions; then we will show the ablation analysis of the InCa algorithm.

4.1 Main Inline Casing Algorithms

Before we show the statistical analysis of the main inline casing algorithms, let us look at the difference between them in the examples presented in the table 4.1. It shows the results of pre-processing and tokenization of the input line, with the flags marked in blue. Firstly, the no-preprocessing system does no casing normalization, thus we can see the upper- and title-cased words being tokenized as they are. As for pre-processing algorithms, we can see the difference between standard InCa and all other algorithms (including naive-InCa): it does not explicitly put the flags whenever it meets a cased word – this is a result of using the vocabulary that was created during training, which allows it to omit putting the flags into the words that are more often cased. In Marian inline casing, we can see with the example of the words "ISIS" or "Martellyho" that it allows to merge the casing flags to the words themselves if they are met frequently. Finally, in the TokenMonster example we can see that it allows for creating the multi-word tokens such as "je to" at the beginning of the second line.

Below we compare the main inline casing algorithms that were proposed by us or by other authors, namely, standard InCa (`inca`), its naive implementation (`inca-n`), marian inline casing function for SentencePiece (`marian`), and capcodes for TokenMonster – `tkm` (we also provide the no-preprocessing system, `base`, for comparison). In all cases, the SentencePiece tokenizer with no specific parameters was used for tokenization. The exception of the TokenMonster with its own tokenizer. We will first compare their performance on the extrinsic metrics on the general (not noised) `flo` development set, and then address their behavior in the noised scenarios.

From the tables 4.2 and 4.3 for the Czech-Ukrainian and Ukrainian-Czech translation directions we can see that all systems including the no-preprocessing implementation go on par, with 21.5-22.0 BLEU score (0.86-0.87 COMET score) variation for Czech-Ukrainian direction and 22.7-23.3 BLEU score (0.86-0.87 COMET score) interval for Ukrainian-Czech. This, as was shown in the previous chapter, also fits within the variation range of 1 BLEU score that we saw for the results of various runs of the same system.

Similar processes can be seen in most of the noise experiments. For example, the variation in fully lower-cased datasets for the Czech-Ukrainian pair is within 18.7-19.2 BLEU points (19.6-20.4 for Ukrainian-Czech), the variation in texts with random casing of 10% tokens is 19.9-21.0 BLEU points (for Czech-Ukrainian) and 21.2-22.5 BLEU points (for Ukrainian-Czech). The detailed overview of the metrics for these types of noising is presented in Appendix A.1.

Preprocessing	**Examples**
Input	Turecko převezme ostrahu bojovníků ISIS Je to Martellyho pátá CEP
base	_Turecko _pře ve z me _o stra hu _bojovník ů _IS IS _Je _to _Mar tel ly ho _pá tá _ CEP
inca	_turecko _pře vez me _o stra hu _bojovník ů _i sis _je _to _ T _mar tel ly ho _pá tá _ cep
inca-n	_ T _turecko _pře vez me _o stra hu _bojovník ů _ U _i sis _ T _je _to _ T _mar tel ly ho _ pát á _ U _cep
marian	T tureck o_ pře vezme _ o stra hu _ bojovník ů_ Uis is T je_ to_ Tmar tel ly ho_ pát á_ Uce p_
tkm	T turecko pře vezme ostr ah u bojovník ů U i sis T [je to] T mar tel ly ho pá tá U ce p

Table 4.1: Illustration on the work of different preprocessing algorithms (and tokenizers applied to the pre-processed texts) with respect to casing. The first row shows the examples of the input sentence; the next rows show the results of the tokenization (the boundary between tokens is shown with a white space). The flags that were used at the pre-processing step are marked blue. The flag T means title-casing of the next word, the flag U upper-cases the whole word. There are other flags that are not represented in the example (for instance, fully upper-cased line or upper-cased spans) that will be shown later. For TokenMonster preprocessing, the square brackets show that the multi-word token boundaries. The preprocessing algorithms' naming follows 2.3.

Prepro-cessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
base	21.6	22.1	51.3	51.8	0.869
inca	21.7	22.4	51.4	52.1	0.870
inca-n	21.9	22.4	51.4	52.0	0.872
marian	21.9	22.5	51.7	52.2	0.876
tkm	21.4	21.9	51.1	51.6	0.870

Table 4.2: Extrinsic metrics comparison of the main inline casing algorithms for the general (not noised) `flo` development set, Czech-Ukrainian translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

Preprocessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
base	22.7	23.2	51.0	51.5	0.873
inca	22.7	23.3	51.0	51.7	0.867
inca-n	23.2	23.7	51.2	51.8	0.873
marian	23.3	23.7	51.4	51.9	0.875
tkm	22.9	23.3	51.0	51.5	0.870

Table 4.3: Extrinsic metrics comparison of the main inline casing algorithms for the general (not noised) `flo` development set, Ukrainian-Czech translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

Preprocessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
base	1.6	1.9	22.5	23.0	0.448
inca	21.3	21.3	51.3	51.3	0.871
inca-n	20.7	20.7	50.7	50.7	0.867
marian	15.5	20.4	39.1	50.7	0.814
tkm	15.5	17.9	46.6	48.9	0.840

Table 4.4: Extrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Czech-Ukrainian translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

The main takeaway from this observation is that, for a general translation task and for some of the case noising scenarios, all suggested preprocessing approaches show similar performance, which does not significantly exceed the baseline without casing preprocessing. This shows that the solution suggested by us is at least on par with the approaches, which is a minimally satisfying scenario for us. In the following, we will focus on the noising scenarios for which various approaches cope with different efficiency; we will also concentrate on the intrinsic metrics that show the optimality of the encoding provided by various algorithms.

A more interesting comparison can be seen in the fully upper-cased noising scenario. In tables 4.4-4.5 we can see that, firstly, all inline casing algorithms show at least ten-times improvement in the BLEU scores and double of the COMET scores compared to the non-preprocessing scenario. Secondly, we see a significant differentiation between the inline casing algorithms. While the lower-cased versions of the BLEU and COMET metrics in most cases are reaching the performance on the baseline datasets (with the exception of TokenMonster), both metric versions that take casing into account show a better performance of InCa and naive InCa compared to Marian and TokenMonster casing. This means that the main difference between the algorithms is that the Marian and TokenMonster casing-trained systems did not output the upper-case flags for the whole sentences (or all words in the sentences).

Qualitative analysis supports this assumption. Indeed, the outputs of the system trained with Marian inline casing have hundreds of non-upper-cased oc-

Preprocessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
base	1.9	2.5	21.8	23.2	0.419
inca	22.8	22.8	51.3	51.3	0.865
inca-n	22.0	22.0	50.8	50.8	0.861
marian	17.6	22.3	41.4	51.0	0.822
tkm	17.6	19.9	47.4	49.3	0.842

Table 4.5: Extrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Ukrainian-Czech translation pair. The legend follows the corresponding Czech-Ukrainian table above.

currences in the texts, which can be grouped into several clusters. Usually, the upper-cased sequences are interrupted either by the title-case flags or the tokens which do not bear case by default (such as punctuation or numerics). Recall that the principle of the Marian upper-case marking is to put the "all-upper-case" flag in the beginning of the span and to end the span with the "return to lower-case" flag. Thus, the problem seems to be in the moment when the fully upper-cased sequences are interrupted by other cases or non-cased elements, after which there is no flag of opening the upper-case span again.

The upper-case inconsistencies generated by TokenMonster can be explained by its architecture. Recall that it allows one to create tokens that consist of multiple words, and the flag for upper-casing the token is marked before each token. However, the upper-casing function at the post-processing step seems to react to the internal white spaces within the multi-word tokens. For instance, in the TokenMonster-strict model, the same reflexive marker "se" in Czech is both a separate token and a part of the multi-word token "stát se" ("to become"). Thus, in cases where the single-word token is used, it is upper-cased since it is directly prepended by the upper-case flag, while in cases such as with token "stát se", only the first verb is upper-cased and the scope of upper-casing finishes at the "surface" white space sign.

Notably, hallucinations in overgeneralization of the casing other than upper-case can be seen in InCa as well. It works as follows: Despite the fact that the whole sentence is prepended with a special flag meaning the whole line is upper-cased, there can be single occurrences marking a particular token within the sentence upper-cased. However, such hallucinations are considerably rarer in InCa than in other preprocessing systems. For instance, in the translated Ukrainian-Czech fully upper-cased dataset InCa had 70 wrong flags of title-cased words; while for SentencePiece there were 568 occurrences of the title-case flags. Moreover, in InCa this is not reflected in the resulting post-processed text as the sentence is left upper-cased once it sees the flag at the beginning of the sentence.[1]

Examples of such hallucinations and inconsistencies made by pre-processing algorithms are demonstrated in Table 4.6. In Marian inline casing, we can see that the words in the rest of the sentence are lower-cased once there was a "break" of the title-case flag in the previous word; in TokenMonster, the upper-case flag

[1]The only problem of the InCa implementation is that by default we did not include filtering out the hallucination occurrences of such flags, thus they are left within the text as the artifact; still, at the production stage they can be easily deleted by an elementary regular expression.

Preprocessor	**Flags**	**Tokenized Output; Postprocessed Result**
marian	A - upper-case span,	A příští_ týden_ v_ elli s_ parku_ v_ Tjohan ne sburg u,_ kdy_ ...
	T - title-case word	PŘÍŠTÍ TÝDEN V ELLIS PARKU V Johannesburgu, kdy ...
tkm	U - upper-case word,	U nebo U [stát se] U silnější m U soupeř em ...
	[] - multi-word token	NEBO STÁT se SILNĚJŠÍM SOUPEŘEM ...
inca	A - upper-case sentence,	A _v _otázce _politiky _ T _blízké ho _východu ...
	T - title-case word	V OTÁZCE POLITIKY *T* BLÍZKÉHO VÝCHODU

Table 4.6: Examples of flag hallucinations that took place in the Ukrainian-Czech translations of the fully upper-cased texts. Each preprocessor (left column) shows the flags that are used to represent upper-case and other casing operations (central column). For each preprocessor, the rightmost column shows at the first line the tokenized output generated by the MT system and then the final output after the post-processing algorithm.

was applied only to the first word in the multi-word token; in InCa, the result was fully upper-cased but the redundant flag remained in the output.

Thus, we can conclude that the general accuracy and fluency of the translations for all above-mentioned algorithms is comparably equal for all scenarios; however, due to the implementation details of the flag decoding (such as in TokenMonster) or to inconsistencies in the principles of the upper-case flag usage (such as in Marian inline casing), the full upper-casing of the output sentence is more consistent in InCa implementations rather than in the other algorithms.

4.1.1 Intrinsic Analysis

To evaluate the intrinsic features of our preprocessing algorithms, we encoded the input sentences with our preprocessors and tokenizers and evaluated them with the intrinsic metrics. The results of this evaluation on the base (not noised) datasets are demonstrated in tables 4.7-4.8. If we look at the character per token (CPT) ratios, we can see that the worst performance is shown by the naive InCa implementation. We can hypothesize that this happens due to its straightforward way of marking every non-lower casing occurrence with a separate token, which increases the number of separate single-character tokens "linearly", thus lowering the average number of characters per token for the dataset. It is notable that for TokenMonster, whose behavior on marking cases is similar to naive InCa (the upper- and title-case flags are put before each occurrence of the cased word), the statistics differ depending on whether it is applied to the Czech text (in this case it is similar to naive InCa) or to the Ukrainian one (in this case it is similar to the non-preprocessed case). Since the number of flags in both texts is comparable, the only interpretation is that the words in the

Prepro-cessing	CPT	AR	EFF
base	3.973	1238	0.538
inca	3.995	1166	0.522
inca-n	3.592	1042	0.423
marian	4.033	1354	0.554
tkm	3.619	1101	0.492

Table 4.7: Intrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Czech data. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

Prepro-cessing	CPT	AR	EFF
base	4.033	1189	0.516
inca	4.014	1113	0.500
inca-n	3.635	997	0.417
marian	4.197	1301	0.563
tkm	4.062	1336	0.503

Table 4.8: Intrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Ukrainian data. The legend follows the table above.

Ukrainian texts happened to correspond to the longer tokens in the TokenMonster dictionary. The better performance is shown by no-preprocessing and standard InCa algorithms (approximately on the same level). The fact that InCa's CPT does not significantly differ from the no-preprocessing scenario is that the number of the auxiliary flags (and therefore tokens) is minimized in this approach; thus, both the denominator in the CPT formula is only slightly increased. The similar process happens in the Marian's inline casing, which shows up to 0.1 additional character in CPT metric. This shows that frequency-based allocation of a flag at least does not decrease the lengths of the encoded tokens, contrary to the explicit encoding of each cased word.

As for the average rank, we can see that the difference between the systems is more substantial. Again, the worst performance is shown by naive InCa, which is then followed by TokenMonster (in Czech language). After that, the close values of InCa and no-preprocessing systems go; while the biggest value is shown by Marian inline casing (and TokenMonster for Ukrainian). On the one hand, this shows better efficiency of the Marian and TokenMonster preprocessors; however, this may be a result of the way these systems treat casing flags and tokens, in general. Recall that both in Marian inline casing and in TokenMonster capcodes, there is no restriction for including the flags into the combinations with other alphanumeric or punctuation sequences. Thus, we can hypothesize that placement of the flagged tokens (i.e. the ones that at least contain a flag) is scattered around the whole ranked distribution, which may tear the average rank towards a higher number. We can see evidence supporting this claim: for instance, in Czech text, Marian uses 420 tokens that contain a flag; 359 out of them have a higher rank

	none		$\text{rand}_{0.1}$		lower		upper		
Prepro-cessing	AR	R(f)	AR	R(f)	AR	R(f)	AR	R(f)	Vari-ation
base	1238	-	1233	-	1047	-	60	-	1178
inca	1166	T:5 U:39 L:28	1085	T:3 U:4 L:18	1069	L:1	1134	A:3	**97**
inca-n	1042	T:1 U:17	985	T:1 U:4	1193	-	1121	A:3	208
marian	**1354**	T:0 U:1628	**1324**	T:0 U:3 A:2475 -A:2468	**1265**	-	**1213**	T:131 U:17 A:0 -A:14	141
tkm	1101	T:3 U:35	1063	T:1 U:5	1135	-	647	U:0	488

Table 4.9: Average Rank and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Czech texts. The names of noising modes follow 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4. The flags are denoted as follow: "T" stands for title-case, "U" – for upper-casing a word, "A" – for upper-casing the whole sentence (or a span for marian), "-A" – for ending the upper-cased span for marian, "L" – for lower-casing the word. The bold figures mark either the best values over the column; for AR scores higher is better, for "Variation" – lower is better.

than the AR score for the document. Thus, diversifying the allocation of the flag depending on whether it should be a separate token or a part of a character sequence helps to flatten the frequency distribution of the token types.

We compared the systems by their AR score on the base dataset, but how important is the absolute value of an average rank? To answer this, we may have a look at the comparison of the systems with respect to different noising of the validation dataset, which is shown in tables 4.9 and 4.10. We are interested in how stable the AR values are for the same algorithm applied to various noising, and how the it is related to the flag ranks in the tokenized datasets. We hypothesize that the ranks of the flags may influence the average rank, most possibly in the following way: the higher the rank(s) of the flag(s), the lower the average rank score as the mean value skews towards the flags in the head of the token distribution. We can see that the statistics support this claim: for instance, for all preprocessing algorithms the AR score slightly decreases if we move from the non-noised dataset to the one with randomly cased 10% of words, and at the same time the title- and upper-case flags (and lower-case for InCa) move towards the head of the distribution, usually getting top-5 places. This can also be seen at the lower-case scenario, where we observe the difference in trends between the algorithms that have no specific flag for the lower-cased words (naive InCa, Marian inline casing and TokenMonster) and standard InCa, as all the former show improvement on AR compared to the standard dataset; while InCa shows a considerable decline as the lower-case token gets the first rank.

Now that we see that there is at least a correlation between the ranks of

	none		rand$_{0.1}$		lower		upper		
Preprocessing	AR	R(f)	AR	R(f)	AR	R(f)	AR	R(f)	Variation
base	1189	-	1154	-	1024	-	46	-	1143
inca	1113	T:3 U:23 L:25	1041	T:3 U:4 L:19	1034	L:1	1091	A:3	**79**
inca-n	997	T:1 U:16	946	T:1 U:4	1135	-	1069	A:3	188
marian	1301	T:0 U:171 -A:1617	1266	T:0 U:4 A:4012 -A:2466	1235	-	**1181**	T:229 U:22 A:0 -A:15	120
tkm	**1336**	T:2 U:28	**1289**	T:1 U:5	**1383**	-	755	U:0	628

Table 4.10: Average Rank and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Ukrainian texts. The legend conventions follow the table 4.9 on Czech data.

the flags and the AR scores, we can think of the stability of the AR score with respect to the casing noise. We do that by computing the difference between the highest and the lowest score of each system among all noising scenarios. Such a comparison shows that InCa, despite being behind Marian and TokenMonster in most separate scenarios, remains the most stable algorithm with respect to any possible noise. For instance, while TokenMonster is the best-performing algorithm on the Ukrainian data, its variation (especially because of the upper-cased noise) exceeds 600 AR points, the variation of InCa does not exceed 80. We believe that this is an important advantage since it gives better predictability of the intrinsic perfromance if we conducted the experiment on one noise and want to hypothesize about the other types of noise.

The final metric that we used for the comparison, Rényi efficiency, gives least preference to naive InCa preprocessing; it is followed by TokenMonster, and then all other systems including the one without pre-processing. If we take that into context of the noising experiments (tables 4.11-4.12), as we did with AR, we will see the motivation behind that. The performance of the metric seems heavily dependent on the presence and frequency of the flags; and the more (and the oftener) the flags, the less the score of the metric. The clearest examples can be seen on the upper-case noising: no-preprocessing scenario gets the highest scores in the table, while the TokenMonster obtains three times as less score (recall that it marks each upper-cased word occurrence with a token, thus it has the biggest absolute number of flags compared to any other algorithm). We understand that this should not be a fair estimate of the non-preprocessing scenario for the future work, as the quality of this system on the downstream performance was between 1.5 and 2.5 BLEU points total. Analogous trends can be seen if we compare other types of noising: for instance, InCa, being the only algorithm that uses flags in the fully lower-cased scenario (to mark the lower-cased words, for instance, in the beginning of the sentence), shows the lowest performance. This is also seen if

	none		$\text{rand}_{0.1}$		lower		upper	
Prepro-cessing	EFF	R(f)	EFF	R(f)	EFF	R(f)	EFF	R(f)
base	0.538	-	**0.549**	-	0.539	-	**0.658**	-
inca	0.522	T:5 U:39 L:28	0.473	T:3 U:4 L:18	0.444	L:1	0.500	A:3
inca-n	0.423	T:1 U:17	0.391	T:1 U:4	0.527	-	0.488	A:3
marian	**0.554**	T:0 U:1628	0.529	T:0 U:3 A:2475 -A:2468	**0.581**	-	0.551	T:131 U:17 A:0 -A:14
tkm	0.492	U:35 T:3	0.489	U:5 T:1	0.452	-	0.226	U:0

Table 4.11: Rényi efficiency and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Czech texts. The names of noising modes follow 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4. The flags are denoted as follow: "T" stands for title-case, "U" – for upper-casing a word, "A" – for upper-casing the whole sentence (or a span for marian), "-A" – for ending the upper-cased span for marian, "L" – for lower-casing the word. The best (highest) scores for each column are marked bold.

we compare each particular system in various noising setups: for instance, naive InCa gets a lower rank of the upper-case flag in the random 10% casing scenario compared to the standard dataset, and while it is used in the lower-cased scenario without any flags, it gets its maximal score.

Can this be a problem of a particular alpha? We made the comparative graphs to see if the ranking of the systems would differ depending on the alpha value. We sampled alphas from 0 to 10 with 0.2 stride and estimated the Rényi efficiency score for each alpha. Then, we compared the performance of the systems for each noising scenario separately. The result of the evaluation on the Czech data is presented in Figure 4.1 (the Ukrainian data show the same patterns). Here, we firstly see that in the majority of the cases, the scores for each system decrease monotonously and do not change their ranking depending on alpha. We can also see that, while for the non-noised and randomly cased 10% scenarios the worst performance is shown by naive InCa (since it uses more tokens than the "smarter" approaches), in the upper-case scenario TokenMonster goes significantly down as it marks each word with a flag), and in the lower-cased scenario, it is InCa with the lower-case flags that lies below.[2]

[2]It is less clear why TokenMonster also shows bad performance on the fully lower-cased data, as it does not use an explicit lower-case flag there. Most probably it is the result of another special token introduced by TokenMonster, "D" token that handles the deletion of the white space after this token. It is used as a way to handle the fully reversible word separation, but in an opposite logic to SentencePiece: while the latter explicitly marks the white spaces, TokenMonster by default restores white spaces between each of its tokens and then deletes them whenever the special token is used. Thus, the frequent usage of this token may skew Renyi efficiency in this case.

	none		rand$_{0.1}$		lower		upper	
Prepro-cessing	EFF	R(f)	EFF	R(f)	EFF	R(f)	EFF	R(f)
base	0.516	-	0.527	-	0.519	-	**0.678**	-
inca	0.500	T:3 U:23 L:25	0.460	T:3 U:4 L:19	0.442	L:1	0.486	A:3
inca-n	0.417	T:1 U:16	0.388	T:1 U:4	0.504	-	0.473	A:3
marian	**0.563**	T:0 U:171 -A:1617	**0.534**	T:0 U:4 A:4012 -A:2466	**0.602**	-	0.566	T:229 U:22 A:0 -A:15
tkm	0.503	U:28 T:2	0.500	-	0.474	-	0.219	U:0

Table 4.12: Rényi efficiency and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Ukrainian texts. The legend conventions follow the table on Czech data above.

The authors of the approach Zouhar et al. [2023] suggest that the increase in alpha should favor the frequent sequences to be encoded into shorter tokens. We cannot say that our evaluation supports this claim. Instead, we can say that it penalizes the systems that output numerous auxiliary tokens (which, in our case, are predominantly single-character). The only exception here is Marian inline casing that sometimes happens to even outperform the non-preprocessing scenario; this can be interpreted due to the nature of the inline casing flags that can be merged with a word, thus not creating a separate token.

In conclusion, we should say that the Rényi efficiency metric (at least in its classical version) does not favor using the characters that increase the number of separate words (and thus tokens). Thus, if we want to encode the flags separately (this is our aim – to relocate the casing information in an way of creating separate tokens), it is impossible to outperform the zero preprocessing scenario on average since any inline approach to casing would at least slightly increase the length of sentence. The case of Marian encoding shows that we can make it better if we allow the flags to merge with the words; but theoretically this does not seem a perfect solution, since if we create a digraph within a word instead of separating it from the word completely, it would not solve the problem of the possible allocation of the same words with different casings in the vocabulary. Bearing all that in mind, hereinafter we will not take this metric into account, since it showed its practical inaplicability for the case of different preprocessing comparisons.

4.1.2 Vocabulary statistics

As a way to address the internal structure of tokenizers, we can analyze their vocabularies as such. We are interested, first, in whether inline casing helps increase the token length in the tokenizer vocabulary; second, how different types of inline casing help release more space for the unique character sequences rather

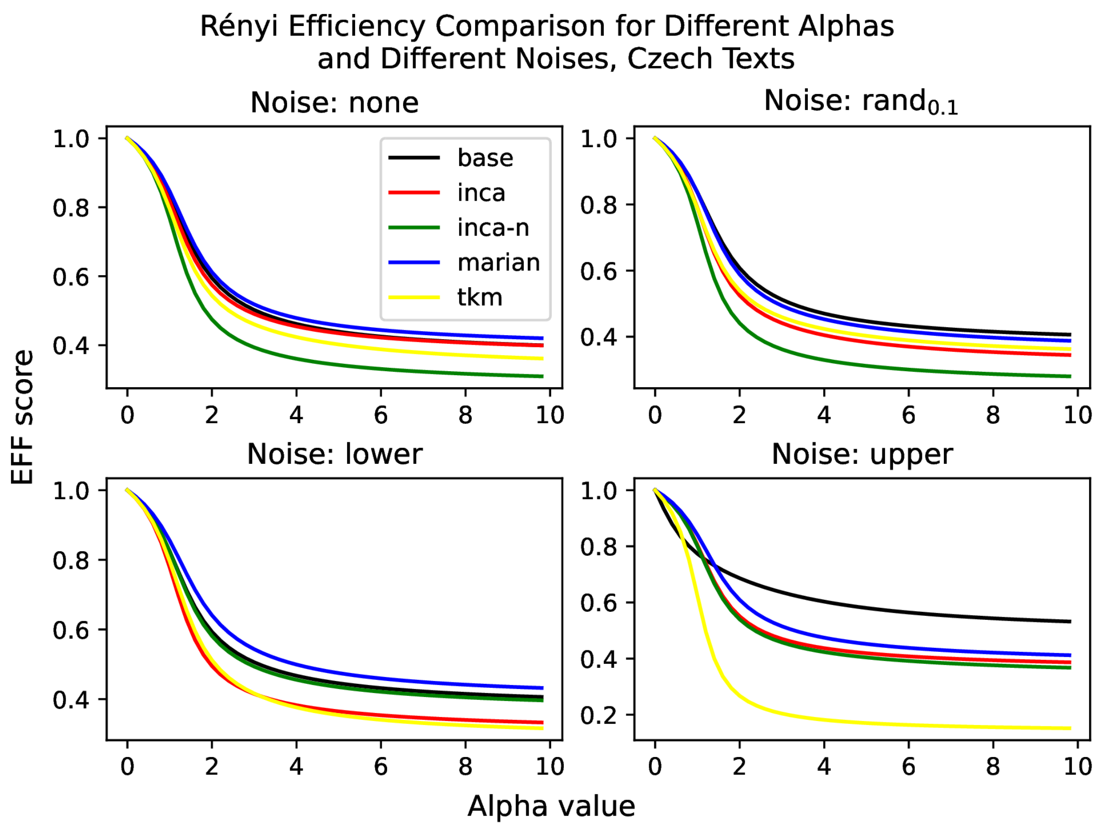

Figure 4.1: Comparison of the Rényi efficiency score depending on alpha. The subplots are created for each type of case noising, each figure shows the EFF score of each system (Y axis) with respect to alpha score (X axis).

than doubling the tokens that differ only in casing. The table 4.13 attempts to estimate that: the CPT$_v$ column provides an answer to the first question, and the "Cased tokens" and "Overlap with Uncased" columns give an estimate for the answer to the second one. We can see that both InCa approaches increase the average unique token length by 0.3 characters. The TokenMonster result is outstanding by increasing the values by more than 1.5 characters; this is one of the features they displayed as their advantage compared to other algorithms, as at their Github they showed that they on the English data the average character per unique token ratio reaches 7 characters. However, this happens due to allowing the tokens to be multi-word ones, which is prohibited by the default SentencePiece implementation: over 6,800 tokens in the Tokenmonster dictionary with an average length of 12.44 characters are present in the tokenizer.

We can also look at how optimal the inline casing approaches are in terms of saving space for unique lower-case character sequences. Contrary to the no-preprocessing scenario where there are more than 6,000 cased tokens, 3,500 of which fully correspond to their lower-cased analogues (which are 19% and 10% of the full vocabulary, respectively), we can see that all inline casing algorithms decrease these numbers several times. [3] However, only the InCa approaches allow us to decrease these numbers to zero, thus allocating all possible space released

[3]It is necessary to note that in case of TokenMonster, all tokens with cased flags are the combinations of the flags with punctuation, not alphanumeric sequences.

Prepro-cessing	CPT_v	Cased Tokens	Overlap with Uncased
base	6.837	6169	3508
inca	7.119	0	0
inca-n	7.127	0	0
marian	6.554	2754	1049
tkm	8.573	149	92

Table 4.13: Vocabulary statistics for the tokenizers trained for the main models. The CPT_v metric is defined in 2.4; the "Cased tokens" value shows the number of tokens that contain a casing flag (or cased letters), and the "Overlap with Uncased" value shows the number of the uncased tokens in the dictionary that differ from the cased ones (in the "Cased tokens") only by casing or casing flag.

by casing normalization to the new tokens. Although this may not be directly reflected in the intrinsic metrics above, this is undoubtedly an important feature for the interpretability and predictability of the tokenizer models, as we expect that the variety of the tokens present in the vocabulary would not be obscured by the casing variation of the tokens.

4.2 Ablation Experiments

4.2.1 Disabling Fully Upper-Cased Strings

One of the standard InCa features is encoding the fully upper-cased lines with only one flag that is prepended to the sentence. We will see how implementation of this feature influences the performance of the system by disabling this method: thus, now at the training step the words from the fully upper-cased lines will be included into InCa vocabulary statistics; and at encoding step each upper-cased word will be (if necessary) encoded with a flag separately. At the flag level it also means that the inventory of the possible flags is reduced by 1: now there is no special flag upper-casing for the whole sentence, only the upper-cased words.

The results show that the general performance of this modification does not differ from the standard InCa – it has approximately the same performance on all translation directions and in all noising scenarios as the standard InCa implementation. The intrinsic metrics do not show any significant difference either. The only difference is the performance on the fully upper-cased noising of the dataset: there the performance on the validation set significantly decreases. This is reflected in Tables 4.14 – 4.15. We have already seen such a trend in other algorithms that do not use a special flag for full sentences (recall tables in A.1 or their excerpts from 4.4-4.5). For instance, in the Czech-Ukrainian direction, the TokenMonster implementation drops from quite a stable BLEU score of approximately 20 to 15.5 in the upper-case scenario and shows 17.9 on lc(BLEU) score. Similar figures are shown with Marian inline casing, as performance drops to 15.5 and 20.4 BLEU and lc(BLEU) scores, respectively. Notably, we can see that with both our ablation and Marian inline casing, the BLEU score is almost completely "restored" to the level of the performance on the non-noised text, while TokenMonster does not show that.

Prepro-cessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
inca	21.3	21.3	51.3	51.3	0.871
inca-A	18.0	18.3	48.7	49.1	0.850

Table 4.14: Extrinsic performance in fully upper-cased scenario, Czech-Ukrainian translation direction. The metrics are formulated as in 2.4. The standard InCa implementation is referred to as “inca” in the “Preprocessing” column, and the ablation without special treatment of the full upper-case sentence is referred to as “inca-A”.

Prepro-cessing	BLEU	lc(BLEU)	chrF	lc(chrF)	COMET
inca	22.8	22.8	51.3	51.3	0.865
inca-A	19.9	20.3	49.3	49.7	0.849

Table 4.15: Extrinsic performance in fully upper-cased scenario, Ukrainian-Czech translation direction. The legend conventions follow the table on Czech-Ukrainian table above.

The closer look at the intrinsic metric shows similar trends in decrease as the TokenMonster performance again. In tables 4.16–4.17 we can see the significant decrease in both CPT and AR values, which resembles the decrease that occurred in the same setup with TokenMonster. Such halving is again logical since both metrics are skewed towards overly frequent upper-case flags, and we can see that if we count the occurrences of the upper-case flags in both encoded documents, in the ablated InCa it is 18799, and in TokenMonster it is 18905.

The analysis provided above shows that introducing the feature that marks large spans of text with upper case (be it a single flag for the whole sentence as in standard InCa or span marker flags as in Marian) is beneficial both for the extrinsic performance on the fully upper-cased texts and for the intrinsic representations of the texts.

4.2.2 Enforcing Flag Separation by SentencePiece

A brief note should be referred to the SentencePiece option user_defined_symbols that allows one to enforce the auxiliary tokens to be separately allocated in the dictionary. This function is initially created for tasks such as “<SYSTEM>” or “<USER>” markers in the dialogue systems, where a whole string must be kept

Prepro-cessing	CPT	AR
inca	3.890	1134
inca-A	1.872	543

Table 4.16: Intrinsic performance in fully upper-cased scenario, encoded Czech text. The metrics are formulated as in 2.4. The standard InCa implementation is referred to as “inca” in the “Preprocessing” column, and the ablation without special treatment of the full upper-case sentence is referred to as “inca-A”.

Prepro-cessing	CPT	AR
inca	3.944	1091
inca-A	1.935	531

Table 4.17: Intrinsic performance in fully upper-cased scenario, encoded Ukrainian text. The legend conventions follow the table on Czech table above.

consistently in all contexts; however, we can use it for the case of our flags. By the definition of our InCa implementation, the InCa flags are expected to be separated from the alphanumeric sequences and punctuation by default. Thus, there should not be any effect on turning on the user_defined_symbols option. We decided to make sure if it is true, or if our preprocessor is working not as optimally, and an explicit separation of the special symbols helps.

The results show that neither extrinsic nor intrinsic metrics show any substantial change compared to the standard InCa implementation; the stable performance was shown on all noising scenarios as well; thus for the sake of brevity, we do not provide any tables to demonstrate that. What is interesting is that the vocabularies trained by the tokenizers slightly differ (the difference tackles only 1,100 tokens out of 32,000). The qualitative analysis did not show any clear trend in the differing subsets of the vocabularies (the average lengths of the tokens and the ratio of the Czech and Ukrainian texts are almost the same there). Thus, we can only hypothesize that the normalization that takes place within SentencePiece tokenizer before the actual training is sensitive to turning on and off the parameter in question, despite the fact that the real training distribution does not differ significantly.

We can conclude from this experiment that the application of the InCa algorithm is self-sufficient and does not require tweaking of the hyperparameters of the tokenizer algorithm. This also allows us to hope that InCa preprocessing should be tokenizer-agnostic; however, we will be able to state this only after comparative experiments with other tokenization algorithms such as BPE or WordPiece.

4.2.3 Casing Augmentation

Training data augmentation is one of the popular and efficient solutions to many problems in NLP. Even specifically with the inline casing problem, the authors of the Marian inline casing Jain et al. [2023] show that their algorithm works best when applied to case-augmented data. Thus, we attempted to see if data augmentation helps the MT system both in no-preprocessing scenario and in pair with InCa.

The data are augmented in the following way: first, the training dataset is multiplied by four, and each instance of the dataset is shuffled. These four copies are noised as follows:

1. data is left as is;

2. data is fully upper-cased;

3. data is fully lower-cased;

4. data is randomly assigned with casing noise with 10% chance.

The upper- and lower-casing of the full data are conducted symmetrically: for each sentence pair, both sentences are upper- or lower-cased. The fourth instance of the dataset, 10% noising, is done asymmetrically – by that we mean that the source and the target sentence are noised separately, thus there is no alignment of the noised words in the sentence pair.

We applied such casing augmentation to both the no-preprocessing scenario and InCa pre-processing. Moreover, for InCa preprocessing, we compared two levels at which augmentation can be applied. In the first case, the augmented data were used both for the InCa vocabulary training and for MT training (which included tokenizer training); while in the second case, the InCa vocabulary was trained only on the initial dataset, while the tokenizer and the MT system were trained on the augmented data. With that, we could better understand the impact of augmentation on the pre-processing algorithm.

The results of the experiments on the non-noised dataset did not show any improvement neither in the non-preprocessing nor in the InCa preprocessing scenario. This resembled the situation with the similar performance of all main algorithms on the default dataset, which we describe in Section 4.1. However, a more interesting observation can be seen in full lower-case and upper-case noising of the validation dataset. The extrinsic evaluation is presented in tables 4.18-4.19. We can see the following trend there: for the fully lower-cased scenario, the breaking point lies between the augmented and non-augmented data, where the difference in BLEU scores yields up to 3 points. However, after augmentation, the difference between the no-preprocessing system and the InCa combinations is within the variation span. In the fully upper-cased data setup, the border (as was discovered earlier) lies between no-preprocessing and InCa systems, and all systems with augmentation do not significantly surpass it.

Does that mean that casing augmentation is a "silver bullet" and we get no improvement from using InCa? To answer this question, we can look at the intrinsic performance of the systems in question (presented in tables 4.20-4.21). We can see that, for both noising setups, the average rank score for the casing-augmented data with no preprocessing is considerably lower than the one in any of InCa setups; while for InCa, casing augmentation does not influence this metric significantly. It is especially clearly marked in the fully upper-cased noising scenario, where the Average Rank for no-preprocessing is multiplied by ten compared to non-augmented data, but is still twice as lower compared to InCas. A similar, however not that clear, trend can be seen with characters per token ratio, where InCa scores remain stable, while case augmentation of the no-preprocessing system either gives 0.5 CPT less (in upper-cased noise) or even slightly decreases the score compared to the not augmented system (in lower-cased noise).

The high performance on the lower-cased data of the non-preprocessing non-augmented system (compared to both augmentation and InCa) can be easily interpreted: since the majority of the tokens in the base system are the lower-cased tokens, applying it to the lower-cased text is the most "comfortable" setup for it. The InCa approach, whether it is augmented or not, will spend additional tokens on marking the lower-case flags where necessary (contrary to fully upper-cased sentence where it only needs one sentence-initial flag); and the non-preprocessing augmented systems were trained in the way they reallocated more

	lower			upper		
Setup	BLEU	chrF	COMET	BLEU	chrF	COMET
base	18.7	49.4	0.849	1.6	22.5	0.448
inca	19.2	50.1	0.856	21.3	51.3	0.871
base+caseaug	22.1	51.8	0.861	21.6	51.4	0.874
inca+caseaug$_{all}$	22.4	52.2	0.864	22.2	52.2	0.877
inca+caseaug$_{MT}$	22.3	51.9	0.862	22.3	51.9	0.875

Table 4.18: Extrinsic performance on casing-augmented data, Czech-Ukrainian translation direction. The metrics are formulated as in 2.4, they show scores for fully lower-cased noising an fully upper-case noising of the development dataset. In the "Setup" column, the non-preprocessing scenario and standard InCa implementation are referred to as "base" and "inca", while "base+caseaug" means casing augmentation for no-preprocessing system, "base+caseraug$_{all}$" stands for casing augmentation applied both for InCa vocabulary and for tokenizer; and "inca+caseaug$_{MT}$" stands for applying casing augmentation applied only on the tokenization and MT step. The horisontal line shows the border between the cluster of the best-scoring systems (differing from each other for less than 1 BLEU point): the rows below are the best cluster and the rows above are considerably worse.

	lower			upper		
Setup	BLEU	chrF	COMET	BLEU	chrF	COMET
base	19.6	49.3	0.847	1.9	21.8	0.419
inca	20.4	50.1	0.853	22.8	51.3	0.865
base+caseaug	23.2	51.7	0.859	22.5	51.4	0.868
inca+caseaug$_{all}$	23.1	51.5	0.856	22.9	51.5	0.867
inca+caseaug$_{MT}$	23.2	51.7	0.857	23.4	51.7	0.866

Table 4.19: Extrinsic performance on casing-augmented data, Ukrainian-Czech translation direction. The legend conventions follow the table on Czech-Ukrainian table above.

	lower		upper	
Setup	CPT	AR	CPT	AR
base	3.924	1047	1.625	60
inca	3.671	1069	3.890	1134
base+caseaug	3.791	940	3.370	610
inca+caseaug$_{all}$	3.695	1073	3.881	1128
inca+caseaug$_{MT}$	3.636	1023	3.852	1084

Table 4.20: Intrinsic metrics on casing-augmented data, Czech text. The metrics are formulated as in 2.4, they show scores for fully lower-cased noising an fully upper-case noising of the development dataset. The values of the "Setup" column are equal to the ones defined in 4.18.

	lower		upper	
Setup	CPT	AR	CPT	AR
base	4.010	1024	1.569	46
inca	3.739	1034	3.944	1091
base+caseaug	3.846	885	3.420	572
inca+caseaug$_{all}$	3.755	1033	3.930	1082
inca+caseaug$_{MT}$	3.695	977	3.894	1029

Table 4.21: Intrinsic metrics on casing-augmented data, Ukrainian text. The legend conventions follow the Czech table above.

Setup	CPT$_v$
base	6.837
inca	7.119
base+caseaug	6.495
inca+caseaug$_{all}$	7.126
inca+caseaug$_{MT}$	7.205

Table 4.22: Average unique token length in the tokenizers with and without casing augmentation.

of their capacities to the cased versions of the words.

The latter claim can be supported by the analysis of tokenizer dictionaries. Firstly, if we compare the average token length in the tokenizers depending on the casing augmentation (see table 4.22 below), we can see that for non-preprocessing scenario the casing-augmented tokens became almost 0.5 characters shorter. At the same time, there is no such drop in tokenizers trained after the InCa application. Moreover, if we look at the details of the tokenizer vocabularies, we will see that for the case-augmented no-preprocessing tokenizer, 12,270 tokens are not lower-cased, and 10,700 of them have their full lower-cased analogues in the vocabulary. This is a demonstration of non-optimal allocation of the vocabulary, contrary to all InCa tokenizers. Regarding the comparison of the tokenizers trained after InCa with and without case augmentation, we will see that the main difference lies between the inca+caseaug$_{MT}$ setup and two other systems, as it has 6500 differing tokens compared to both other alternatives. The qualitative analysis did not show any pattern in the differing tokens: in all cases, there is a similar amount of proper nouns (which are expected to be written with the title case) and the ratio between the Czech and Ukrainian sequences.

As a result of this ablation experiment we can see that, on the one hand, casing augmentation is a straightforward and efficient algorithm that can improve the extrinsic performance of the MT even without preprocessing. However, the optimality of the tokenizer (which can be seen through its vocabulary or through the intrinsic metrics) is significantly reduced. However, with InCa setups, augmentation does not provide any significant growth in extrinsic metrics; but it shows stability in both intrinsic and extrinsic performance showing that we can even manage without augmentation with no significant performance loss. The only counter-example to the InCa performance is the difference in the augmented and non-augmented lower-cased noise; however, we can hope that it can be solved by a simple modification of InCa by adding the full-sentence lower-case flag, analogously to the full-sentence upper-case flag. Due to the lack of time, we did not conduct such an experiment and leave it to future research.

4.3 Intrinsic Comparison of Tokenizers Trained on Varying InCa Vocabulary Size

The crucial feature of the standard InCa pre-processing method is to create a vocabulary that would store information about the most frequent casing of the words met in the training corpus. However, if we record all words that occurred in the training data and were predominantly cased, the resulting dictionary, according to Zipf's law, will likely consist of numerous hapax legomena or words that occurred only a few times. On the one hand, this may be beneficial as we store more information about the possible rare proper names; on the other hand, it makes the algorithm vulnerable to single occurrences. Thus, we introduced a training parameter responsible for the minimal count of the cased occurrences of a particular word to be recorded into the vocabulary.

In the extrinsic experiments described above, the InCa took minimal count as 1, thus including any occurrence of the cased word if it was more frequent than the non-cased one (even in case if it was the only occurrence of the word). The naive implementation of InCa, although technically implemented slightly differently to speed up the algorithm, essentially can be thought of as a standard InCa implementation with a minimal count set to infinity (thus the InCa dictionary will always be equal to zero). We decided to take a deeper look at the distribution of the cased tokens that InCa was taking into account, and to conduct several experiments with intrinsic evaluation to see if a particular parameter of the minimal counts would influence the intrinsic metrics of InCa.

To begin with, we calculated the distribution of the tokens in the standard InCa dictionary with respect to their counts in the training data. We wanted to estimate how many upper- or title-cased tokens were met once, twice, and other number of times. The results of this calculation are represented in Figure 4.2. The x-axis of the graph represents the counts of a particular unique word in the training data that was recorded in the InCa dictionary. The y-axis shows how many words with a given count (on the x-axis) are in the training data. For instance, if there were three unique words that were met 1,000 times, it would correspond to a point with the x-coordinate 1000 and y-coordinate 3.

From the figure, we can see, firstly, that the number of the upper-cased tokens is approximately one order of magnitude lower than that of the title-cased tokens. Secondly, both upper- and title-cased tokens are ordered in the log-log distribution, thus there is indeed a big imbalance towards the words that occurred once or a few times. Thus, we decided to create the InCa vocabulary with minimal counts equal to logarithmically increasing values, namely, 1 (standard), 5, 10, 50, 100, 500, 1000 and 5000 occurrences, as well the naive implementation that does not store any words (which, according to the plot above, roughly corresponds to minimal counts of 10,000). We created the InCa dictionaries, trained the tokenizers, and applied them to our flores dataset and its noised forms.

The results of the intrinsic evaluation of the documents tokenized with differently pre-processed data are presented in the figure 4.3. We can see the following

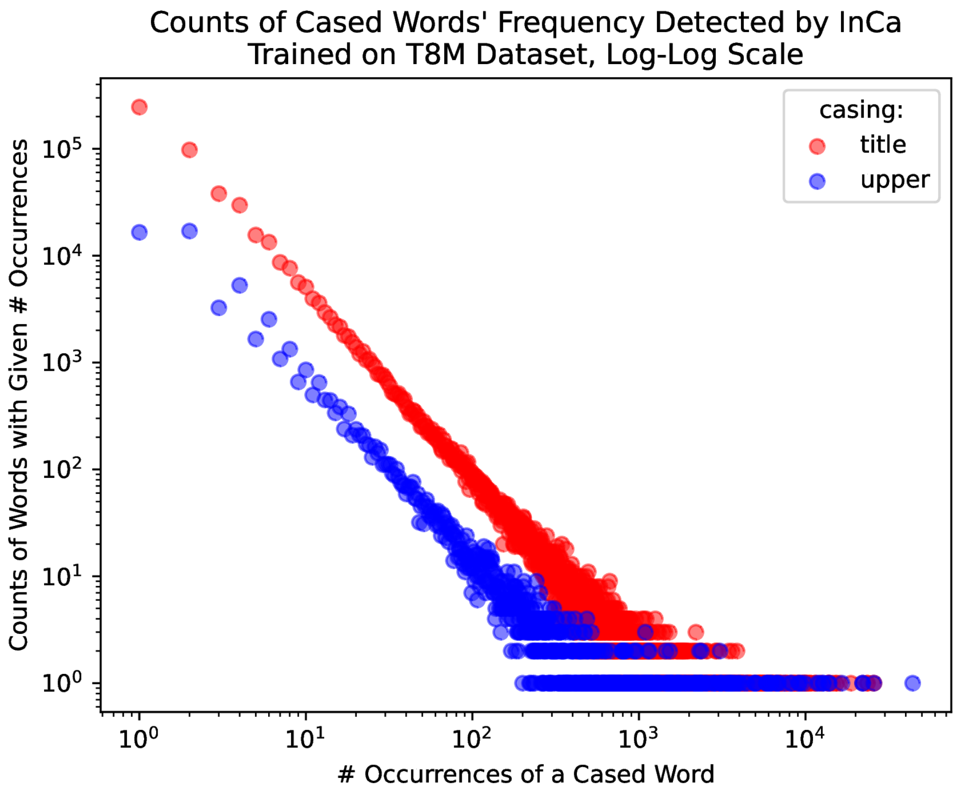

Figure 4.2: Counts of the number of unique words that were met in the corpus a particular number of times. The upper-cased and the title-cased words are marked separately. The x-axis represents the number of times a particular unique word was met in the training data. The y-axis represents how much words were met a given number of times. Both axes are logarithmic.

general trends: the lower-cased text encoding efficiency gradually increases with the increase of the minimal counts; while the non-noised and randomly cased 10% words encoding lose their efficiency. The range of the values for CPT is approximately 0.5 charaters and for AR – over 100 ranks. In particular, the dynamics are similar for both languages and for both metrics. The increase in lower-case performance is easy to interpret: When we decrease the number of the cased words that we store in the InCa dictionary, each of their lower-cased occurrences does not have to be marked with lower-case flag. At the same time, each cased word that is now not in the InCa dictionary has to be marked with an upper- or title-case flag from now on. This is also supported if we specifically analyze the flag ranks: for instance, for the no-noised scenario on the minimal count range of 1 to 1000, the title-case flag rank increases from 5 to 3, the upper-case flag rank increases from 39 to 27, and the lower-case flag rank drops from 28 to 62. It is also notable that all graphs show two intersection points – between the fully lower-cased and random 10% noise (approximately at the minimal count of 10), as well as between the no-noising, fully lower-cased noise and almost with upper-cased noise (between the minimal counts of 1000 and 5000).

What are the words that are being filtered out at every new threshold? The words of count 1 to 5 usually comprise much junk that emerged as an artifact of web crawling (for instance, “новиниКонцепт” which is literally translated as

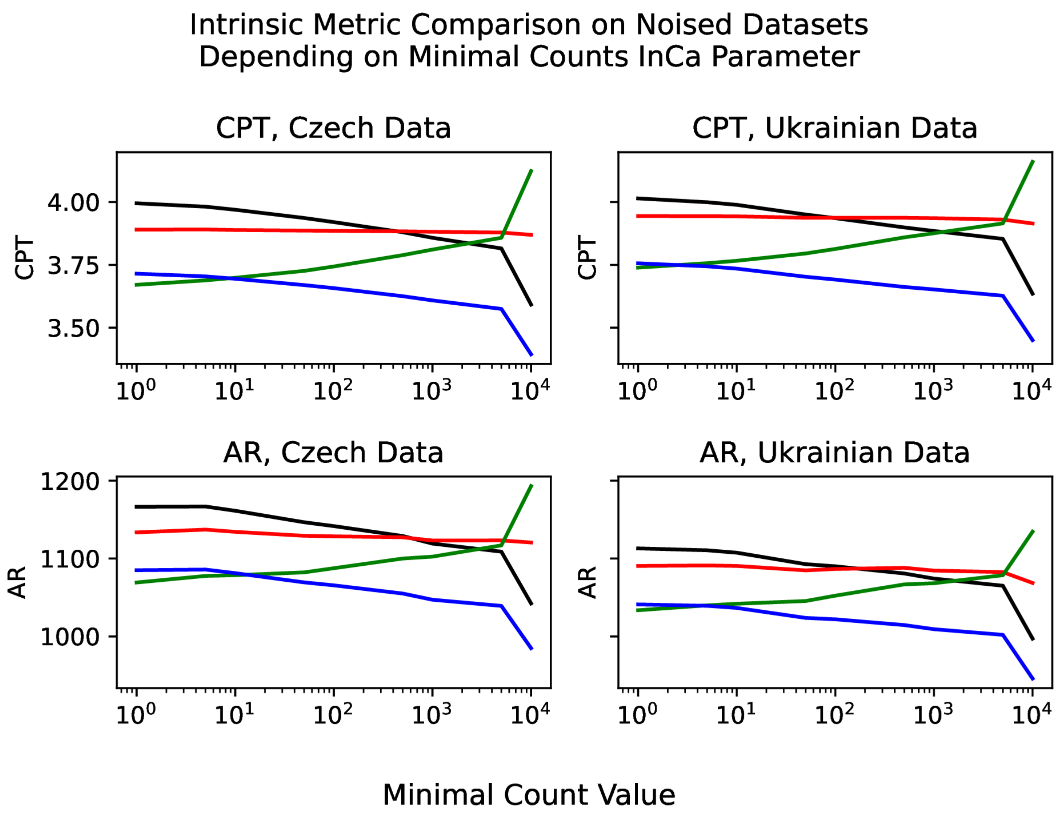

Figure 4.3: Comparison of the intrinsic metrics for the tokenizers trained on InCa vocabularies with different minimal counts. The first row shows the character per token ratio; the second one – the Average Rank ratio (defined in 2.4). The left column shows the performance on the Czech data, the right – on the Ukrainian data. Each graph shows the efficiency of the metric (y-axis) with respect to minimal counts value (x-axis, logarithmic). On each graph, four lines represent different noising of the encoded dataset: black for no noising ("none"), red for fully upper-cased; green for fully lower-cased; blue for random 10% casing.

"newsConcept" is most probably a wrongly deciphered HTML line which combined two separate words), the words from languages other than Czech, Ukrainian or English (for instance, the Portuguese "Libertação"), and the technical abbreviations like "FM1100". At the threshold of 10, junk starts to disappear and numerous international acronyms start to appear (such as ICMPv6 for "Internet Control Message Protocol for IPv6"). At thresholds of 50 and 100, we can see a steep increase in the human names, especially in the indirect cases (such as "Chruščova", "Навального" ("Navalny" in genitive), "Tutanchamona" or "Даніеля" ("Daniel" in genitive)), as well as locations and words derived from it (such as "Karlsruhe" or "Страсбург" meaning the city of Strasbourg). When the threshold is 500, we start obtaining names of the popular companies or cultural entities (such as "Nintendo", "Pinterest" or "Євангелії" meaning "Gospel") and names of the countries or regions (like "Chorvatsku" or "Польщу" meaning Poland). The levels of 1000 words do not break this trend; however, it is interesting to notice that even there we can find numerous occurrences of the English cased words such as "Palace" or "Tower"). Thus, since there seems to be a switch from the unfrequent (and often junked) occurrences of the cased words towards the frequent namings of the popular personalities, companies, or locations, it may be a valid solution to introduce a threshold between 50 to 500 words.

However, the InCa implementation is language-agnostic and does not take into account the word declination (which is the case for both Czech and Ukrainian languages). Should we worry about the inconsistency in the declination paradigm of the same lemma? In other words, should we worry that the nominative case "Česko" would be recorded in the dictionary but the genitive case of the same word,"Česka", would not? The preliminary qualitative analysis shows that, if we take a particular proper name and look at the counts of its forms with respect to grammatical cases, the counts of the forms would usually be within the same order of magnitude. This seems to be consistent for both very frequent and moderately rare entities. For example, the counts of the grammatical forms of the lemma "Rusko" span from 2600 to 10900 (with 8,000 to 9,000 occurrences on average); while the counts of a rarer toponym such as "Alžírsko" span from 107 to 218 (with the only exception of the instrumental case with 21 occurrences). Thus, we can hope that establishing a threshold would not make a situation of inconsistency of the case marking with respect to grammatical forms frequent, because all forms of the particular word would tend to either be recorded or not.

Thus, based on our intrinsic and qualitative overview, we can suggest that the optimal minimal counts value should be approximately 100. At that point, the values of non-noised and 10% randomly-noised words do not decrease yet, while the lower-cased encoding starts increasing; and at the same time, the vocabulary becomes free of junk or the occurrences of very infrequent words.

5. Experiments with Diacritization Strategies

This chapter addresses experiments with the inline diacritization approaches that we presented in Section 2.2. Since we have not found any existing solution of that type, we will compare the two main InDia versions – Char-InDia and Word-InDia – with the base (no-preprocessing) system first; then, we will address the ablation experiments with Char-InDia.

Before we delve into the quantitative analysis of the results, let us have a look at how two main inline diacritization results look like after pre-processing and tokenization. The example of this is represented in the table 5.1. Recall that both approaches use the pre-trained vocabulary (similar to standard InCa), but they differ in marking the less frequent diacritizations for each base (bare sequence of the Latin characters of the word). In case of Char-InDia, the flag consists of a dictionary of key-value pairs, where each key is an index of a diacritized character and the value is an ID of the diacritization operation; the morphology of the flag is $KV - idx_1 - ID - idx_2 - KV - d_1 - d_2$, where a special symbol KV separates the sequences of keys (in the beginning) and values (in the end), and a special symbol ID separates the indices of the diacritized characters (which are marked by numbers). This can be a long sequence, but each diacritization symbol consistently bears information about its operation, and each key deterministically shows the position of the diacritized symbol. In the case of Word-InDia, all diacritizations of the same base are ordered at the training step, and during inference, each diacritization which is not most frequent is assigned with its rank in the ordered list of the particular base. This is shorter, but each flag loses its semantics.

In the illustration, we can see that both char-InDia and word-InDia omit flags on the word "Olympijské", since it is stored in their dictionaries. We also see that in case where the word is non-diacritized while the most frequent version of its base is diacritized, they both use flags that erase diacritization (in the case of the word "komisi", for which the most frequent diacritization is "komisí"). We can see that, in case of the non-diacritized word, the words which have the diacritization different from the most frequent one tend to be over-tokenized by the no-preprocessing system, while they are kept as a whole in both InDia setups (such as the word "stálá"). Finally, we can see that if we disregard the flags, the tokenization of the bases for each word is the same with Word-and Char-InDia.

5.1 Main Inline Diacritization Algorithms

Below, we will compare solutions for three of our systems: without preprocessing (base), character-level InDia that uses the vocabulary to store the most frequent diacritics and uses sequences of flags for each diacritic sign in a word (char-InDia), and a word-level InDia that uses the vocabulary with all diacritizations stored and uses single flags for each word to show the ID of the whole diacritization sequence for a particular base.

Apart from the translation comparison on the default flores dataset, we also

Preprocessing	**Examples**
input	Olympijské komisi Spojených států stálá tajemnice Nobelovy komise
base	_Olymp ijské _kom is i _Spojených _států _stál á _tajemn ice _Nobelov y _komise
char-InDia	_Olymp i jske _KV 5 KV n _komisi _Spojenych _KV 4 KV k _statu _KV 2 ID 4 KV č č _stala _ta jem nice _Nobelov y _komise
word-InDia	_Olymp i jske _ N _komisi _Spojenych _ 1 _statu 2 _stala _ta jem nice _Nobelov y _komise

Table 5.1: Illustration of two inline diacritization methods applied to the Czech excerpts. Two first rows show the inputs, the next lines show the results of pre-processing and tokenization ("Base" means no pre-processing). The InDia flags are marked in blue. For char-InDia, KV flag marks the separator between the keys (indices of the diacritized character) and values (flags for each character diacritization), and ID is a separator if there are multiple keys. k means putting "kroužek" diacritization, č means putting "čárka" diacritization, n means de-diacritizing the letter. For word-InDia, N means de-diacritizing the whole word, 1 and 2 mean the second- and the third- frequent diacritizations for the same base. The preprocessing algorithms' naming follows 2.3.

see if the system is robust against the texts with deleted diacritics (either completely or in 20% words). Since Czech is considerably richer in diacritizations than Ukrainian, we only use the Czech-Ukrainian translation pair for the noising scenario. However, for the default flores dataset, we use both translation directions to demonstrate that the system can be used on the target side as well.

The table 5.2 shows the extrinsic performance on noise in the non-noised setup for both directions. We can see that both Char- and Word-InDia handle both translation directions well and result in very similar scores. It is especially important for the Ukrainian-Czech translation direction, as it shows both the ability of the MT system to learn the token sequences which contain flags, and the InDia decoder allows one to correctly restore the diacritics in the resulting files.

As for the noised scenario (presented in table 5.3), we can see that both InDia approaches handle the task significantly better, doubling the quality on the fully de-diacritized text and yielding 3 BLEU points in the 20% de-diacritized text. It is notable, though, that for the fully de-diacritized scenario, the performance of Char-InDia is stably lower. Since it lies within 1 BLEU point span, thus this may be a matter of stability of the NMT training; however, this may be a consequence of how the de-diacritization is marked in two approaches. We will see the demonstration of this below, when we look at the intrinsic performance.

	Czech-Ukrainian			Ukrainian-Czech		
Prepro-cessing	BLEU	chrF	COMET	BLEU	chrF	COMET
base	21.6	51.3	0.869	22.7	51.0	0.873
char-InDia	21.7	51.5	0.872	22.8	51.0	0.867
word-InDia	21.7	51.3	0.867	22.7	51.1	0.870

Table 5.2: Extrinsic metrics comparison of the main inline diacritization algorithms for the general (not noised) `flo` development set, Czech-Ukrainian and Ukrainian-Czech translation pairs. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

Noise	Preprocessing	BLEU	chrF	COMET
strip	base	9.2	36.0	0.552
strip	char-InDia	17.9	47.2	0.812
strip	word-InDia	18.8	49.1	0.827
$\text{strip}_{0.2}$	base	18.6	48.4	0.800
$\text{strip}_{0.2}$	char-InDia	21.1	50.9	0.862
$\text{strip}_{0.2}$	word-InDia	21.1	51.0	0.861

Table 5.3: Extrinsic metrics comparison of the main inline diacritization algorithms for the noised `flo` development set, Czech-Ukrainian translation direction. The noising naming follows 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

Noise	Preprocessing	CPT	AR
none	base	3.973	1238
none	char-InDia	3.709	1112
none	word-InDia	3.903	1156
strip	base	2.829	418
strip	char-InDia	1.604	480
strip	word-InDia	2.530	742
$\text{strip}_{0.2}$	base	3.675	1110
$\text{strip}_{0.2}$	char-InDia	2.922	878
$\text{strip}_{0.2}$	word-InDia	3.511	1040

Table 5.4: Intrinsic metrics comparison of the main inline diacritization algorithms for the un-noised and noised `flo` development set, Czech texts. The noising naming follows 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4

.

5.1.1 Intrinsic Evaluation of the Czech Texts

We also compared the intrinsic metrics of our systems in Czech texts (since we do not apply InDia to the Ukrainian side). We can see from the table 5.4, that, for both non-noised and noised scenarios, the no-preprocessing and Word-InDia metrics show similar performance, while Char-InDia shows significantly worse scores. This is, again, logical due to the type of noise we are examining (only de-diacritization) and the way it is handled by both approaches. As for many words, there has to be a de-diacritization flag, Word-InDia handles it in a more optimal way by assigning a single-character token, as Char-InDia can produce long sequences representing all character IDs which should be de-diacritized. This makes it similar to the poor intrinsic performance on the fully upper-cased data of TokenMonster, where it had to apply the upper-cased flag before each token, and thus was intrinsically very inefficient.

What is more surprising is why the no-preprocessing and Word-InDia algorithms show similar intrinsic performance. The reason for that seems to be lying in the way the de-diacritized text is tokenized in the no-preprocessing and in Word-InDia systems. In the first case, the word is over-tokenized since it cannot find the corresponding diacritized substrings, and because of that it uses more tokens for a given sentence. In the case of Word-InDia, the words are not over-tokenized, but almost every word has a flag showing that it is de-diacritized. Thus, two essentially different processes have the same effect of decreasing the intrinsic metrics values. The example of this over-segmentation (for the no-preprocessing system) and over-use of the flags (for Word-InDia and Char-InDia) is shown in the table 5.5. We can see that the de-diacritized word segmentation of the bases (pure Latin character sequences) in both InDia implementations is reasonable; however, they are overwhelmed with the non-diacritization flags.

5.1.2 Vocabulary Analysis

One of our hopes during the inline diacritization implementation was that the tokenizer would allocate more space to the longer Latin character substrings via

Preprocessing	**Tokenization Output**
Initial Sentence	Na různých místech v...
De-diacritized	Na ruznych mistech v...
base	_Na _ru z ny ch _mi st ech _v ...
Word-InDia	_Na _ N _ruznych _ N _mistech _v ...
Char-InDia	_Na _KV 1 ID 4 KV NN _ruznych _KV 1 KV N _mistech _v ...

Table 5.5: An example of the fully de-diacritized text tokenized by no-preprocessing (base) and two InDia algorithms. At the first two rows, the initial (diacritized) and the de-diacritized input are shown. In the InDia outputs, the blue characters represent the flags, out of which the N means stripping diactirization (for Word-InDia – full word, for Char-InDia – for particular character). For Char-InDia, KV shows the separation between the key-value parts of the flag, and the ID is a delimiter of the indices of the characters that should be diacritized in the word.

Preprocessing	CPT_v
base	6.837
char-InDia	6.919
word-InDia	6.911

Table 5.6: Average unique token length in the SentencePiece vocabularies with and without inline diacritization. The metric CPT_v is defined in 2.4.

allocation of all diacritization information to flags. The table 5.6 shows that it indeed increased the average unique token length by almost 0.1 characters. Still, the increase is not substantial (it is three times smaller than it was with the inline casing). This can be explained if we look at the diacritized subwords in the no-preprocessing tokenizer. There, out of 32,000 tokens, we will see 8,454 diacritized subwords (which comprises a quarter of the whole vocabulary and approximately a half of the Czech subwords there), but only 583 were having a non-diacritized analogue. This fundamentally differs from the trends in the inline casing-optimized vocabularies described in 4.1.2, where up to a half of the cased unique subwords are doubled with the non-cased ones. Thus, despite helping having more consistent word splitting with respect to de-diacritization noise, the potential for increasing the lengths in the non-cased vocabularies is very restricted.

We also analyzed the difference between the Char-InDia and Word-InDia vocabularies. The overlap between them reaches 31,000 tokens, which shows that the exact choice of inline diacritization does not significantly influence the creation of the bare Latin tokens.

5.1.3 Analysis of Char-InDia Flags

Since Char-InDia allows for the multi-character flags due to its marking of each diacritized character, we allowed the tokenizer to combine the flags for separate character diacritization operations. In 2.2.2, we hypothesized that combining such flags may even show some linguistic patterns within the Czech language. In

Cluster	Inputs	Outputs
hč	žádáni	KV 0 ID 1 ID 5 KV hč n zadani
	washingtonští	KV 10 ID 12 KV hč washingtonsti
	končí	KV 3 ID 4 KV hč konci
hh	měď	KV 1 ID 2 KV hh med
	višňových	KV 2 ID 3 ID 6 KV hh n visnovych
	vyproštovaly	KV 5 ID 6 KV hh vyprostovaly
nč	pojmenovaná	KV 8 ID 10 KV hč pojmenovana
	napsaný	KV 4 ID 6 KV hč napsany
	drahý	KV 2 ID 4 KV hč drahy

Table 5.7: Examples of the clusters of the Char-InDia diacritization operations flags that are generalized by the SentencePiece tokenizer and seen in the encoded flores dataset. The h flag means applying háček, č flag means čárka, n flag means de-diacritization of a character. The flags KV and ID are auxiliary tokens that delimiter the indices of the diacritized flags and their operations, the numbers in blue show the indices of the characters (0-based) to which the diacritization operations should be applied.

the following, we will provide a short analysis of whether it was true. To begin with, the range of the diacritization operations that we allowed for Char-InDia was 4 (operations for marking a letter with "háček", "čárka" or "kroužek", plus de-diacritization of the character). The SentencePiece tokenizer that was trained on the data preprocessed in this way generated 14 tokens that contained one or more diacritization flags. Four of them were representing the single flags, and ten others either had combinations of de-diacritization (2 or even 3 de-diacritized flags as one token) or the combinations of the diacritization operations (sometimes together with de-diacritization). Thus, the idea of generalizing the diacritization operations into clusters indeed has some potential. On the other hand, we see that the most frequently combined operation is de-diacritization, which rhymes with our analysis on the performance on de-diacritized texts and the need to possibly create a flag that would signify de-diacritization of all characters in a word.

However, do these diacritization clusters represent some clear features of the Czech language? We cannot speculate on that based on the tokenizer vocabulary only, thus we looked at the text tokenized with Char-InDia and analyzed the words that were assigned the flags with the combinations of the diacritization operations. Such occurrences were not abundant, but some combinations, such as applying two háček signs or a háček and a čárka did occur tens of times in the texts. Examples of such occurrences are shown in the table 5.7 below.

We could not see the obvious linguistic patterns in any of the diacritization clusters. As can be seen from the table, they can relate to various parts of speech and denote different phonemes standing at different distances from each other. The only two interesting observations that may give a hint of the underlying patterns are the following. Firstly, we can see that the háček+háček combinations seem to be tied to the neighboring consonants (such as št). Secondly, the combination of de-diacritization+čárka tend to be assigned to adjectives or participles. Apart from the examples in the table, there are also occurrences of such word classes that relate to the same tokenization, for instance, “nazyvaná” or “radá”. These are anecdotal evidence of such trends and we cannot talk about that confidently, but we believe that there is space for analysis of that on the bigger datasets and in different setups.

5.1.4 Analysis of Hallucinations in Ukrainian-Czech MT Output

As for applying char-InDia on the target side, we performed an analysis of the hallucination counts in the output texts. Despite all other approaches covered in the work, be it InCa or Word-InDia, the Char-InDia approach is distinctive for its use of the sequences of characters that are used to denote a flag for one word. Moreover, according to the principles of SentencePiece training, we are sure that these combined flags are represented as sequences of tokens (because of different classes of signs used to separate the keys from the values in the flag). Thus, we should be more concerned about hallucinations in this scenario than in the others, since long sequences of tokens may be more vulnerable to that. We estimated two types of the most obvious hallucinations: firstly, the pointers to the non-existent character IDs in the next word (for instance, a pointer to make a “háček” on the seventh letter of the word, although the length of the word is 3); secondly, the incompatible diacritic-character combination (for instance, a “háček” pointer on the letter “p”, which is not permitted in Czech). We will call the first type of hallucinations “out-of-range”, and the second “wrong sign” type.

However, the results of the MT output are encouraging. Out of 19,760 words in the target text (detokenized after output), there are 642 char-InDia flags, and only 8 of them show hallucinations (5 of out-of-range type, 3 of wrong sign type). What is even more interesting is that if we calculate the hallucinations at the validation set during the model training, it will show that this small number of hallucinations (summing to approximately 10) becomes stable by the end of the first epoch. Thus, it seems that for the standard Char-InDia implementation, such combined flags are easily processed, and we should not fear the combinations of the tokens that represent a single flag.

5.2 Ablation Experiments

5.2.1 Data Augmentation

We tried augmenting the training data to estimate if enhancing the system with fully or partially de-diacritized sentences will help its performance. For augmentation, we used the following method: we created quadruples of the training data, within which the initial (diacritized) Czech data were kept intact two times, fully de-diacritized data were introduced once and partially de-diacritizded (in 20% of the words) data were introduced once. The target Ukrainian data were always intact.

We compared the performance in the non-augmented and augmented scenarios of both non-preprocessing system and Char-InDia. Since we expect the Ukrainian data to be translated into diacritized text, we only tried the noising experiments in the Czech-Ukrainian direction. The performance on the non-noised dataset

	strip			$\text{strip}_{0.2}$		
Setup	BLEU	chrF	COMET	BLEU	chrF	COMET
base	9.2	36.0	0.552	18.6	48.4	0.800
char-InDia	17.9	47.2	0.812	21.1	50.9	0.862
base+Aug	21.2	51.0	0.857	21.6	51.4	0.869
char-InDia+Aug	21.2	50.8	0.856	21.4	51.3	0.867

Table 5.8: Extrinsic performance on diacritization-augmented data, Czech-Ukrainian translation. The metrics are formulated as in 2.4, they show scores for fully de-diacritized and 20% de-diacritization noising of the development dataset. The "Setup" naming denotes the systems the following way: "base" and "char-InDia" stand for the no-preprocessing and Char-InDia systems trained on initial data; "+Aug" suffix means the training data was augmented with de-diacritized texts as described above. The horizontal line between the metrics shows the border between the group of the best scoring results (below the line) and the results that are at least 1 BLEU point worse (above the line).

	strip		$\text{strip}_{0.2}$	
Setup	CPT	AR	CPT	AR
base	2.829	418	3.675	1110
char-InDia	1.604	480	2.922	878
base+Aug	**3.727**	**884**	**3.841**	**1275**
char-InDia+Aug	1.618	482	2.927	875

Table 5.9: Intrinsic performance on diacritization-augmented data, Czech texts. We compare the fully de-diacritized and 20% de-diacritization noising of the development dataset. The system naming conventions follow the extrinsic table 5.8 above. The figures marked bold show the best results in each parameter and setup.

was comparable for all systems, and thus we do not describe it here in detail. It was more interesting to compare the noised scenarios both from the extrinsic and the intrinsic perspectives. The results of this comparison are shown in Tables 5.8-5.9.

We can see that data augmentation, indeed, helps to improve the quality of the translation and the efficiency in the intrinsic metrics and allows to reach the scores for the non-noised scenarios. As for the extrinsic metrics, we can see that for the fully de-diacritized texts, the border lies between augmentation and non-augmentation, while for the partial de-diacritization, the standard Char-InDia already shows a similar performance. This is explained by the fact that, for the partial de-diacritization, the method of de-diacritization encoding of Char-InDia still allows us to keep all information without generating too long strings; while for the full de-diacritization, the long de-diacritization flags start being a problem for the MT systems.

The evidence supporting this can be found in the intrinsic metrics table, as we see that Char-InDia suffers from a significant drop in the fully de-diacritized scenario, and it persists even in the scenario of data augmentation. What is even more impressive, is that we can see the only significant increase in the internal metrics with the no-preprocessing and data augmentation setup.

Does this mean that we should give up InDia in favor of data augmentation for de-diacritization noise? Firstly, we can see the potential in Char-InDia: as was already seen from the comparison of the main algorithms (Word- and Char-InDia), the problem of the solely character-level inline diacritization is that for the fully de-diacritized texts it makes the encoded lengths too long, and a simple trick such as marking the word with one "un-diacritized" flag (as is done with word-InDia) helps both intrinsic and extrinsic performance. Thus, it can be easily modified in the future implementations of InDia to make the performance on the fully de-diacritized scenarios better. What is also seen from the data is that in the partial noising even the current Char-InDia is already a working instrument, which does not require augmentation. Moreover, we can see the stability of the intrinsic performance of the algorithm, similar to what we have seen in InCa ablation studies. On the one hand, it is sad that the augmentation does not improve it; on the other hand, it shows that we do not have to think of tuning the augmentation and can rely on the initial data.

Secondly, we should also look at the tokenizer vocabularies trained on the no-preprocessing data for the augmented and non-augmented dataset. Recall that in the non-augmented scenario, the share of the diacritized tokens there was over 8,000, and only 500 of these tokens had their de-diacritized pairs in the vocabulary. In the augmented scenario, the number of the diacritized tokens is reduced to 6162 tokens, but the number of its de-diacritized pairs increased to 2976, yielding almost 10% of the whole vocabulary. This shows that in the augmented data scenario, the the capacities of the tokenizer trained with no preprocessing are not optimally used, and in a way are used even less efficiently than in no augmentation scenario (since more diacritized words have their de-diacritized pairs). InDia approach solves the problem of such overlap, but to make it more efficient in terms of intrinsic and extrinsic metrics of the texts, we will have to combine the word- and char-level methods in one solution.

5.2.2 Naive Char-InDia

In Section 2.2.2 where we were presenting Char-InDia, we argued that, when assigning the word with diacritization flags, we should depart not from the base (non-diacritized Latin character sequence), but from its most frequent diacritized form. We decided to verify this claim by running the extrinsic and intrinsic evaluation of the Czech-Ukrainian translation on the "naive" implementation of Char-InDia. In this scenario, analogously to naive InCa, we do not track the most frequent diacritizations of each word; instead we depart from the base and mark every diacritization explicitly with the Char-InDia technique. The results of this evaluation in comparison to the standard Char-InDia are shown below in the tables 5.10-5.11.

The experiments support our initial claims: using the naive implementation does not help us improving extrinsic performance on the initial data and partially de-diacritized data (the ranges of variation in chrF and COMET even give us hints that this can be statistically significant difference; but we will not claim that as we know about the instability of the training systems). It is also seen in the intrinsic metrics, as the values of CPT and AR drop to their halves. The only slight increase of performance is seen on the fully de-diacritized noise, which

Noise	Setup	BLEU	chrF	COMET
none	Char-InDia	21.7	51.5	0.872
none	Char-InDia-n	21.0	50.4	0.855
strip	Char-InDia	17.9	47.2	0.812
strip	Char-InDia-n	18.4	48.0	0.804
$\text{strip}_{0.2}$	Char-InDia	21.1	50.9	0.862
$\text{strip}_{0.2}$	Char-InDia-n	20.5	50.1	0.849

Table 5.10: Extrinsic performance comparison of standard Char-InDia and naive Char-India (marked "Char-InDia-n"), Czech-Ukrainian translation. The metrics are formulated as in 2.4, each two lines show comparison on no noising, full de-diacritization and 20% de-diacritization of the development dataset (defined in "Noise" column).

Noise	Setup	CPT	AR
none	Char-InDia	3.709	1112
none	Char-InDia-n	1.579	478
strip	Char-InDia	1.604	480
strip	Char-InDia-n	4.092	1207
$\text{strip}_{0.2}$	Char-InDia	2.922	878
$\text{strip}_{0.2}$	Char-InDia-n	1.807	545

Table 5.11: Intrinsic performance comparison of standard Char-InDia and naive Char-India (marked "Char-InDia-n"), Czech texts. The system naming conventions follow the extrinsic table 5.10 above.

is logical, as the naive implementation does not use any flags on the fully de-diacritized text. Still, even in this setup we see that the extrinsic performance does not reach the non-noised results of the translation.

Finally, if we look at the tokenizer vocabulary trained on the naive Char-InDia, we will see that the number of tokens consisting of only the diacritization flags has increased to 31 (from 14 in standard Char-InDia), which is again less optimal. Having all this in mind, we can conclude that our decision to use the most frequent diacritization of the word was indeed a better idea than to use the base.

6. Experiments with Romanization

The final chapter of our experimental research is related to the effect of Ukrainian romanization on both the Czech-Ukrainian translation and on the intrinsic metrics of encoding Ukrainian and Czech texts. Firstly, we will look at the extrinsic performance of Czech-to-Ukrainian and Ukrainian-to-Czech translation, and then we will analyze if romanization has a positive impact on the encoding of the Ukrainian texts. We compared only the no-preprocessing scenario for both languages, to evaluate the effect of sole romanization on both directions. We compare two types of romanization presented in 2.2.3 and the scenario without romanization, where we treat Ukrainian as Cyrillic. Recall that the difference between the romanization types is treating the soft sign, which is initially (marked "roman" in the table) not switched to a Latin character due to the absence of its analogue; however, this appears to enforce token splitting over this character because of the SentencePiece restriction on consistent Unicode script within the same token. The modified romanization, called "roman$_{+soft}$", switches the soft sign to an auxiliary Latin character as well.

The results of the translation evaluation in both directions are represented in Table 6.1. We can see, similarly to the inline casing or inline diacritization, that our romanization techniques do not lose in performance in both directions when applied to the Ukrainian. We also do not see any substantial difference between the two romanization variants.

We will now take a closer look at the intrinsic performance of the romanization techniques, presented in Table 6.2. If we look at the encoding performance of Ukrainian texts, we will see an improvement of up to 0.2 characters per token and up to 100 ranks in the average rank score. We can also see that the complete romanization (which includes the soft sign) works better than that using the Cyrillic soft sign. This is evident since SentencePiece consistently split the words over that sign. This can be seen in detail if we look at the tokenizer vocabularies generated for each system. In the "roman" case, the only token containing the soft sign is the soft sign itself. In the no-preprocessing vocabulary, however, we see that there are over 1700 tokens containing the soft sign; thus the necessity in romanizing all cheracters is clear. In "roman$_{+soft}$" we see better handling of the soft sign, as there are 1057 tokens containing the soft sign. We should also note

	Czech-Ukrainian			Ukrainian-Czech		
Romanization	BLEU	chrF	COMET	BLEU	chrF	COMET
none	21.6	51.3	0.869	22.7	51.0	0.873
roman	21.7	51.4	0.870	23.0	51.2	0.874
roman$_{+soft}$	21.5	51.3	0.872	22.8	51.1	0.872

Table 6.1: Extrinsic performance comparison of no preprocessing and two romanization preprocessing techniques for Czech-Ukrainian and Ukrainian-Czech translation directions. The metrics are formulated as in 2.4. In column "Romanization", "none" stands for the "base" experiments without preprocessing, "roman" stands for romanization of all characters except for the soft sign, and "roman$_{+soft}$" stands for romanization of all characters including the soft sign.

	Czech		Ukrainian	
Romanization	CPT	AR	CPT	AR
none	3.973	1238	4.033	1189
roman	4.065	1328	4.095	1223
roman$_{+soft}$	4.049	1320	4.261	1286

Table 6.2: Intrinsic performance comparison of no preprocessing and two romanization preprocessing techniques for Czech and Ukrainian texts. The system naming conventions follow the extrinsic table 6.1 above.

Romanization	CPT$_v$
none	6.837
roman	7.071
roman$_{+soft}$	7.134

Table 6.3: Average unique token length in the SentencePiece tokenizer vocabularies for no preprocessing and two romanization preprocessing techniques. The system naming conventions follow the extrinsic table 6.1 above.

that implementing the romanization helped to increase the intrinsic vocabulary metric: as Table 6.3 shows, there is an increase in the CPT$_v$ metric for both types of romanization, and romanization with soft sign performs the best.

6.1 Vocabulary Overlap Estimation

Our main goal was to increase the overlap between the token coverage of the two related languages. Did we succeed in that? If we look at the results of tokenization (for instance, at the table 6.4), we will easily see that in many cases, the token overlap was granted due to a simple latinization of the Ukrainian (it works for both loanwords like "Tokio" and Slavic cognates like "bude"), and at the same time many words that are obviously linguistic cognates differ slightly and because of that cannot be mapped to the same tokens (such as "jedynym" and "jediným"). Thus, we will need an estimation of how successful we were.

We used two approaches to estimate the token overlap. We took the corresponding tokenized texts in Czech and Ukrainian and, firstly, counted the overlap of the unique tokens in the two texts. We also computed the number of the unique tokens in both encoded texts and obtained the intersection-over-union score, showing the fraction of the shared unique tokens to the total of the observed unique tokens. Secondly, we calculated the probability distributions of the tokens for both texts and applied the Jensen-Shannon distance metric to these distributions. Contrary to the intersection over union, the Jensen-Shannon distance takes into account the frequencies of the tokens, thus it should be less sensitive towards rare occurrences of the corresponding tokens in two texts (for instance, if the same English word was used once in two texts). The results of this comparison are presented below in the table 6.5.

We can see that for both romanization approaches, the overlap jumped significantly to over 1700 tokens, yielding 5% of the whole 32,000 subword dictionary and to 13% of the tokens used in the particular texts. The JSD metric also de-

Romanization	Ukrainian	Czech
input	Токіо буде єдиним азіатським містом,	Tokio bude jediným asijským městem,
none	_Токіо _буде _єдиним _азіатськ им _містом ,	To ki o _bude _jediným _asijský m _městem ,
roman	_Tokio _bude _jedynym _aziats ь kym _mistom ,	_Tokio _bude _jediným _asijský m _městem ,
roman$_{+soft}$	_Tokio _bude _jedynym _aziatsk ym _mistom ,	_Tokio _bude _jediným _asijský m _městem ,

Table 6.4: Illustration of the romanization experiments on the encoded Czech and Ukrainian sentence. The first line shows the input sentence before (possibly romanization and) tokenization. The overlapping tokens in the two languages are marked blue. The system naming conventions follow the extrinsic table 6.1 above.

Romanization	Overlap	IoU	JSD
none	285	0.020	0.780
roman	1751	0.130	0.630
roman$_{+soft}$	1738	0.129	0.627

Table 6.5: The degree of overlap in the encoded Czech and Ukrainian bitext. The "Overlap" column stands for the count of unique tokens met in both texts (bigger is better), "IoU" stands for Intersection-over-Union score (fraction of overlap value by the number of unique tokens used in either of the texts), and "JSD" stands for Jensen-Shannon Distance (scale 0-1, lower is better). The system naming conventions follow the extrinsic table 6.1 above.

creased by 0.15, which is a considerable change bearing in mind that even the noised versions of the texts in the same language have a high JSD: for instance, the non-noised and lower-cased versions of the file in the same language have a JSD score of 0.19, and the non-noised and de-diacritized versions – 0.45.

The last comparison that we conducted was the estimation of the generalization potential of SentencePiece training on texts with the same writing system. We took the SentencePiece dictionary from the initial (no-romanization) setup, where we found 15,027 out of 32,000 tokens that consisted of the Cyrillic characters. We took them all and romanized them straightforwardly with our romanization script. Then we searched in the SentencePiece vocabulary that was trained on the romanized data to find the complete analogues of the initial Cyrillic tokens that we romanized with a script post factum. We could find 13,394 such tokens. This gives us a hint that for most romanized tokens in Ukrainian, their distribution is still independent from the distribution of the Czech tokens, therefore most of them are grouped the same way with no regard to the alphabet they are encoded with.

From the comparisons conducted above we can see that a straightforward romanization of the Ukrainian characters (or, in case of the palatalized consonants, character bigrams) allows us to increase the overlap between the tokens both in the tokenizer vocabulary representations and in the token distributions observed in the tokenized texts of the two languages. Still, we see much space for improvement with respect to both trying the inline algorithms described in the earlier chapters, as well as more elaborate versions of mapping the Cyrillic texts on the Latin script.

7. Comparison of Intrinsic and Extrinsic Metrics

We have already examined the comparison of the extrinsic and intrinsic performance of various algorithms within the particular setups. In this chapter, we will give a broader overview of the metric distributions and see if there is correlation between the intrinsic and intrinsic performance of tokenization.

Firstly, we will take a look at the metrics on all experiments and visualize the paired distributions of the main intrinsic (CPT, AR) and extrinsic (BLEU, chrF, COMET) metrics used in the analysis. The graph representing this visualization is shown in the figure 7.1. We can see that, in general, a slight linear correlation between the extrinsic metrics and intrinsic metrics takes place, and in all intrinsic-extrinsic pairs it is even statistically significant (with a p-value lower than 0.05). Judging by Pearson and Spearman's correlation coefficients, the average rank statistic is more correlated with the extrinsic values than the character per token ratio; out of the extrinsic metrics, the BLEU and COMET scores tend to be slightly more correlating than the ones of chrF.

However, from the visualization on the graphs we can see two common problems for all combinations: on the one hand, for the intrinsic scores of the "medium" quality, there are multiple cases of similar intrinsic values corresponding to a large scale of the extrinsic values. This makes such a comparison unreliable for the cases that represent the systems which are in the middle of the distribution. Another problem is that for the best scoring extrinsic systems (within a very small range of variation over the y-axis), the variation in the intrinsic metrics is very wide. This makes intrinsic metrics unhelpful if we want to distinguish similar but highly scoring systems, which is the crucial case for MT in general.

Comparing all setups within one plot and one statistical test may blur the patterns that can possibly be seen within a particular text. This is especially important for the potential use of the intrinsic metric as a way to search for the optimal configurations for the extrinsic systems. For that task, we are mostly interested in how the relative ranking of intrinsic metrics within a given text reflects the ranking of the resulting extrinsic performance. Thus, we conduct separate ranking tests for each language and for each noising scenario. We used both the Spearman rank correlation coefficient (in table 7.1) and Kendall tau (in table 7.2).

We can see from this comparison that indeed some of the noising parameters (although only related to casing) start showing stronger ranking correlation between the extrinsic and intrinsic metrics. This is not surprising for the fully upper-cased scenarios (where the correlation coefficients are the highest), as we could see from the detailed analyses. However, we see that for the randomly cased 10% of words scenario the correlation is also considerably high. Another interesting observation is the case of the fully lower-cased noise: here, the coefficient is negative, which may be surprising as both our intrinsic and extrinsic metrics are organized by the "higher is better" principle. If we look at the systems with the highest intrinsic (Marian inline casing, TokenMonster) and extrinsic (various types of casing augmentation, InCa) scores for this scenario, we may see a pos-

		CPT against...			AR against...		
Lang	Noise	BLEU	chrF	COMET	BLEU	chrF	COMET
cs	none	0.215	0.346	0.105	0.083	0.182	0.172
cs	lower	-0.468	-0.594*	-0.424	-0.224	-0.365	-0.194
cs	upper	0.668*	0.676*	0.656*	0.603*	0.626*	0.594*
cs	$\text{rand}_{0.1}$	0.594*	0.516*	0.488*	0.571*	0.504*	0.482*
cs	strip	0.214	0.262	-0.048	0.619	0.667	0.429
cs	$\text{strip}_{0.2}$	0.333	0.405	0.214	0.143	0.19	0.048
uk	none	0.281	0.055	0.21	0.227	-0.064	0.15
uk	lower	-0.715*	-0.77*	-0.556*	-0.571*	-0.626*	-0.506*
uk	upper	0.735*	0.691*	0.629*	0.738*	0.688*	0.591*
uk	$\text{rand}_{0.1}$	-0.213	-0.344	-0.293	-0.253	-0.379	-0.287

Table 7.1: Spearman's rank correlation between the intrinsic metrics (denoted above) and the extrinsic metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The metrics are formulated as in 2.4. The statistically significant scores with p-value lower than 0.05 are marked with an asterisk.

		CPT against...			AR against...		
Lang	Noise	BLEU	chrF	COMET	BLEU	chrF	COMET
cs	none	0.13	0.282	0.06	0.028	0.147	0.108
cs	lower	-0.283	-0.367	-0.276	-0.117	-0.233	-0.109
cs	upper	0.483*	0.5*	0.467*	0.433*	0.45*	0.417*
cs	$\text{rand}_{0.1}$	0.384*	0.304*	0.246	0.355*	0.275	0.246
cs	strip	0.071	0.143	0.071	0.429	0.5	0.429
cs	$\text{strip}_{0.2}$	0.286	0.357	0.143	0.071	0.143	-0.071
uk	none	0.194	0.036	0.151	0.166	-0.04	0.091
uk	lower	-0.477*	-0.561*	-0.427*	-0.433*	-0.5*	-0.4*
uk	upper	0.583*	0.55*	0.45*	0.583*	0.55*	0.45*
uk	$\text{rand}_{0.1}$	-0.142	-0.192	-0.151	-0.15	-0.233	-0.159

Table 7.2: Kendall's tau correlation between the intrinsic metrics (denoted above) and the extrinsic metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The statistically significant scores with p-value lower than 0.05 are marked with an asterisk.

		CPT against...			
Lang	Noise	BLEU	lc(BLEU)	chrF	lc(chrF)
cs	none	0.178	0.39	0.291	0.461*
cs	lower	-0.468	-0.379	-0.594*	-0.646*
cs	upper	0.668*	0.721*	0.676*	0.75*
cs	$\text{rand}_{0.1}$	0.511*	0.469*	0.407	0.488*
uk	none	0.241	0.124	-0.019	-0.241
uk	lower	-0.715*	-0.711*	-0.77*	-0.758*
uk	upper	0.735*	0.815*	0.691*	0.797*
uk	$\text{rand}_{0.1}$	-0.213	-0.368	-0.344	-0.312

Table 7.3: Spearman's rank correlation between the CPT metric (denoted above) and the standard and lower-cased versions of BLEU and chrF metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The metrics are formulated as in 2.4. The statistically significant scores with p-value lower than 0.05 are marked with an asterisk.

sible explanation to it: the pre-processing algorithms in the former group favour the lower-cased data (e.g. they use flags for all casing apart from lower-case), while the algorithms in the latter group are trained for the general robustness in various tasks. This leads us to the conclusion that we should pay attention to what are the "preferred" options of noise for each pre-processing system, because if they are over-sensitive to a particular noising, they may start optimizing the intrinsic performance while losing in the extrinsic scores.

Finally, we should pay special attention to absence of correlation in the diacritization noise. This can be explained by the internal structure of the flags that are used in Char-InDia, as they themselves can consist of multiple characters and be even split to multiple tokens. This shows that for various types of typographic effects, different intrinsic metrics can be helpful. What is also important is that the versions of the extrinsic metrics, if they are prepared for a particular type of noising, can show bigger correlation with the given intrinsic metrics. This can be shown in Table 7.3, where we compare the intrinsic metrics and the standard BLEU and chrF scores with their lower-cased versions (for the sake of brevity, we only show an excerpt with the comparison of CPT against the extrinsic metrics; with AR, however, the trends are similar). We can see that the correlation scores that were seen with standard BLEU and chrF, are consistently higher in the lower-cased versions of these metrics; moreover, we see more statistically significant correlations which even tackle the non-noised scenarios.

All the observations above show us that for different types of typographic phenomena (casing or diacritization), both extrinsic and intrinsic metrics should be carefully chosen and paired. This motivates us for future research in finding a better way of intrinsic evaluation, especially for the de-diacritized texts.

7.1 Intrinsic Metrics Comparison

We have seen that we can benefit from using the intrinsic metrics for estimating the extrinsic performance on our tasks (although only in a small subset of scenarios). But how important is it to use both character per token ratio and average rank? To answer that, we plot the distribution of two intrinsic metrics and compute their correlation. The results are shown in the figure 7.2. We can see the obvious correlation from the visualized distribution, supported by the Pearson's and Spearman scores. Since we have not seen an obvious advantage of any of the intrinsic metrics compared to each other in the section above while compared directly to the extrinsic metrics, and since the intrinsic metrics highly correlate between each other, using both metrics does not add much information to intrepreting the models performance and we can resort to one of the two.

The last metric that we have to compare is the average length of a vocabulary item, CPT_v. We are interested in seeing whether this metric that is related to the internal structure of the tokenizer can be a good predictor for the intrinsic or the extrinsic metrics. We created a graph representing the distribution of the CPT_v scores of each experiment and the scores of the extrinsic and intrinsic metrics that were calculated for the texts generated by the tokenizers. The results are represented in the figure 7.3 for one extrinsic and one extrinsic metric only, as the behavior of the other metrics is similar. We can see that, for both BLEU and CPT, the distribution of the metrics is skewed toward the upper left quadrant, and at the same time, the range of the values for each CPT_v score is very wide. Thus, we can conclude that CPT_v should not be considered as a predictor of the extrinsic or intrinsic performance of systems. However, we still find this metric useful as it provides a better understanding of the inner structure of the tokenizer, not only the distributions that it generates.

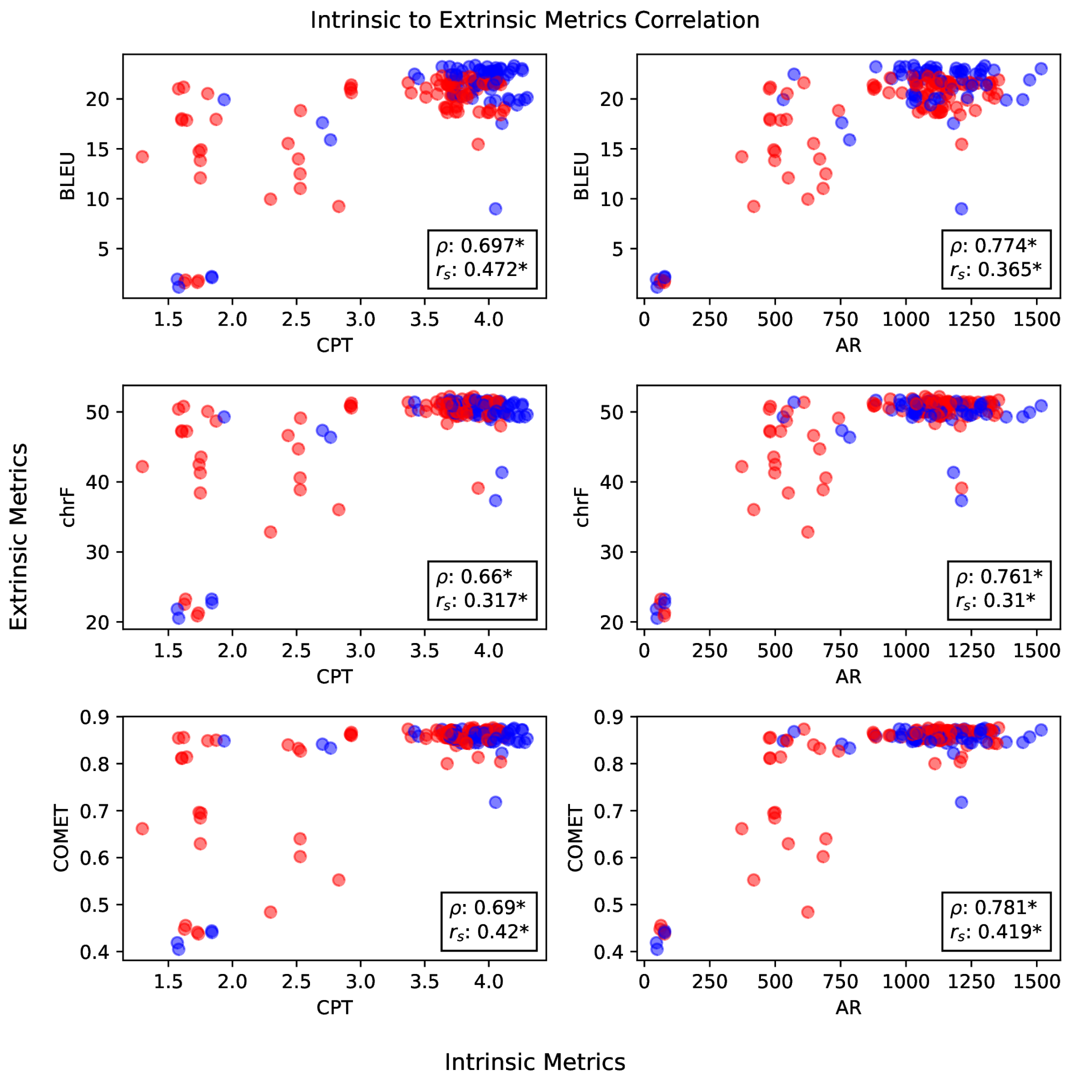

Figure 7.1: Pairwise comparison of the main extrinsic and intrinsic metrics on all experiments. In each graph, the x-axis represents the intrinsic metric score and the y-axis – the extrinsic metric score. The colours represent the language of the encoded text (and, for extrinsic metric, the source) – red is for Czech (and Czech-Ukrainian direction), blue is for Ukrainian (and Ukrainian-Czech direction). The ρ stands for Pearson correlation coefficient, the r_s stands for Spearman's rank correlation coefficient; the asterisk shows if the p-value is lower than 0.05.

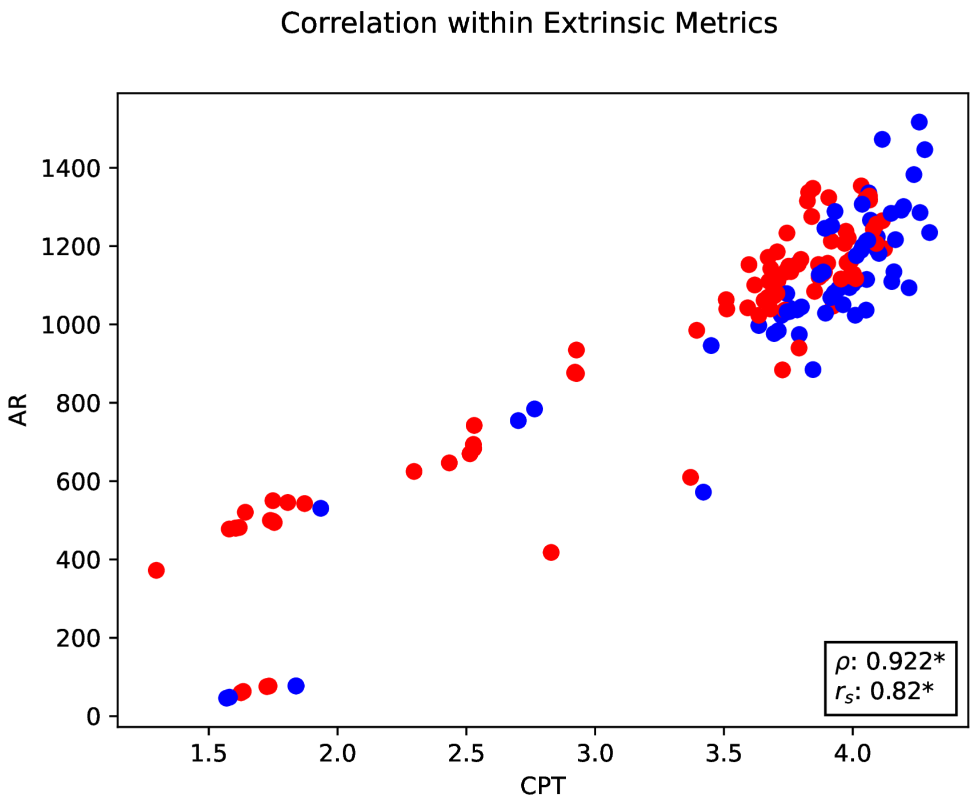

Figure 7.2: Comparison of two intrinsic metric distributions – character per token (CPT, x-axis) and average rank (AR, y-axis). The colours represent the language of the encoded text – red is for Czech, blue is for Ukrainian. The ρ stands for Pearson correlation coefficient, the r_s stands for Spearman's rank correlation coefficient; the asterisk shows if the p-value is lower than 0.05.

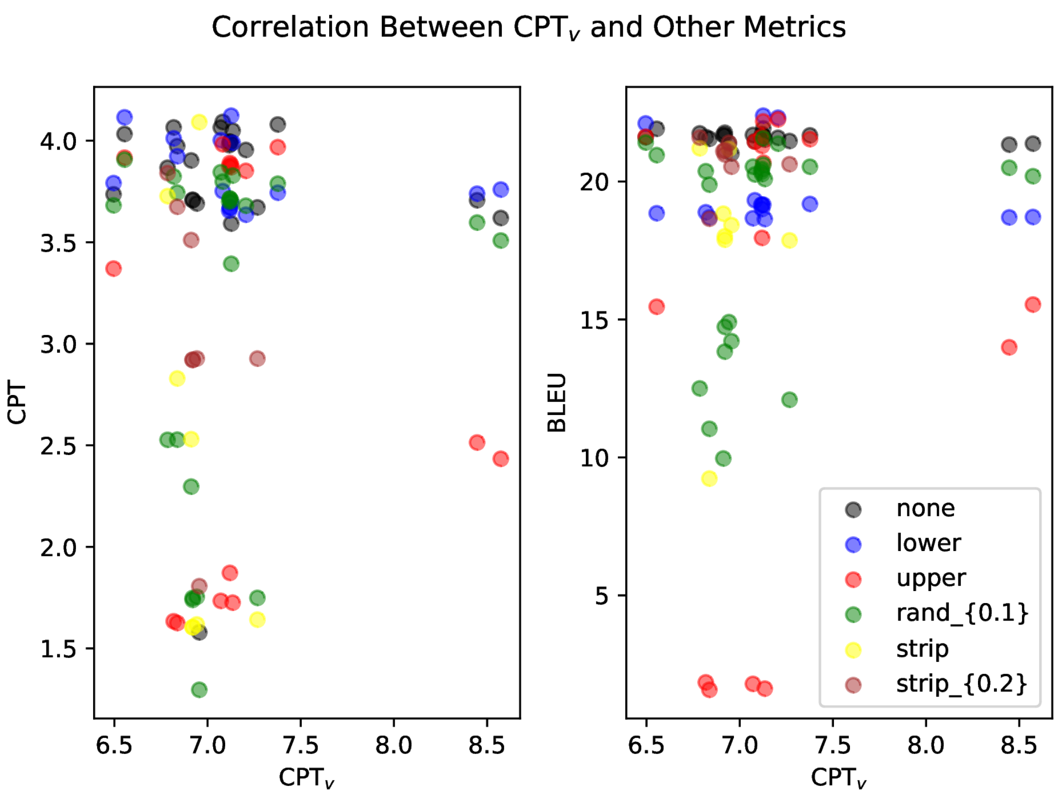

Figure 7.3: Comparison of two intrinsic CPT_v score on the tokenizer and the intrinsic (CPT) and extrinsic (BLEU) scores for the texts generated by the tokenizers. The CPT_v score is shown over x-axis, the extrinsic and intrinsic metrics are shown over y-axis. Each data point shows a metric for a particular text; they differ by the color depending on the type of noising of the text which is encoded.

Conclusion

The primary goal of our work was to introduce the inline casing algorithms that would enhance intrinsic and extrinsic performance in Czech-Ukrainian machine translation in noised scenarios. We show that the proposed inline casing solution, InCa, manages to show the best performance on some of the noised texts while performing on par in other noising scenarios. We also conduct the analysis of the intrinsic performance of our algorithm and the existing alternatives and show that the proposed algorithm is more stable and that the tokenizer vocabulary generated by it is more interpretable than competing solutions. This is a crucial advantage of an algorithm in the era of neural NLP and large language models, since "just good performance" is not enough, as it does not provide the explanation of black-box processes within the system. Moreover, our algorithm does not require data augmentation, which was considered the preferred setup for some of the inline casing algorithms against which we compared our system.

The second goal of our work was to leverage the inline approach for the preprocessing of diacritized texts. Being the first, to our knowledge, to try it in this field, we show that this approach shows significant improvement on the dediacritized Czech texts without losing performance on the original data. We see that two "extremes" of the inline diacritization that we have suggested, character-level InDia and word-level InDia, have imperfections, and the best solution would be to try to combine these two approaches. Unfortunately, due to the lack of time and compute resources, we did not manage to do that within the thesis.

The third goal of our research was to examine the impact of romanization of Ukrainian texts for the Czech-Ukrainian translation pair. We show that the intuition behind increasing the shared vocabulary can be supported by the observations, and the extrinsic performance of the system remains stable. Still, we see that there is space for generalizing the vocabulary even more, although it has to be done through either more fine-grained romanization approaches or through combination of romanization and other preprocessing (for instance, the above-mentioned InCa and InDia).

Finally, we aimed to perform the simultaneous extrinsic and intrinsic examination of the tokenizers with respect to preprocessing, as well as comparing the extrinsic and intrinsic metrics on our data. Within our research, we showed that one of the recently suggested metrics, Rényi efficiency, is not applicable for the setup of the pre-processed tokenizers and we instead use other metrics to explain the internal processes of the tokenizers. We also show that, apart from the interpretation of the tokenizers as they are, the intrinsic metrics tend to correlate with the extrinsic performance in some of the noising scenarios, which may encourage the community to use them as approximations for choosing the optimal tokenizers without training the full rounds of tokenizers with downstream neural NLP systems, which takes much time and compute. At the same time, we show that the scope of this correlation is quite narrow and depends on the particular noising scenario and the choice of the extrinsic metrics used for the evaluation.

Our research has several limitations. Firstly, we restricted the scope of languages to the single language pair of the same family, which uses similar orthographical principles. Even within the European language area and Latin script-based languages, there are other orthography systems such as German, where each noun is title-cased; thus, we cannot claim that the performance and stability of our system will be replicated for other language pairs. Similar problems stand for diacritization, as there are languages that use a significantly wider range of diacritics (such as Vietnamese), for which our InDia system may end up being inefficient.

Secondly, we evaluated our approaches (and, for inline casing, the competing systems) on a single validation dataset whose domain partially overlaps with the training data (Wikipedia). We need to evaluate the robustness of our (and competing) systems towards unseen domains, as our system relies on a pre-trained vocabulary of the words seen in the training data.

Finally, for the sake of comparison of the extrinsic performance of the systems, we did a limited training of the MT systems. For instance, the current version of the Charles translator for Ukraine which is accessible online shows a stable performance of several BLEU points higher than ours, since it uses a bigger Transformer model and is trained for a week (contrary to one day in our case) and on the back-translation augmented data. Thus, we did not claim that our algorithm reaches state-of-the-art on the Czech-Ukrainian translation pair; instead, we fixed all training parameters and compared the performance of various accessible preprocessing approaches within the same setting. The problem of the restricted training setup is also reflected in the extrinsic metrics performance: as we have shown, BLEU scores for the systems that were trained with exactly the same parameters show variation up to 1 BLEU point; this puts constraints on our conclusions that are made about the comparison of the different algorithms that we did. However, we tried to show the intervals of such variations and to pay more attention responsibly to situations where the difference in the extrinsic scores was greater than the expected variation.

Apart from the future experiments that we have already mentioned above to mitigate the limitations of our work (such as applying the systems to other languages or testing them in other domains), there are several other directions for prospective research in this field.

Firstly, the inline approaches suggested by us focus on the word-level variation of the words; however, we can imagine that the casing noise or diacritization omitting will be on particular characters within the word. It would be interesting to compare our approach with other existing approaches for such setups and possibly to enhance InCa for that (while character-level InDia seems to be ready for it, but it needs verification).

Secondly, the tokenizers used in our comparison were both based on the Unigram language model in SentencePiece (and on a similar approach in TokenMonster). Thus, it would be useful to see how our approaches would help the NMT system if the tokenizers trained on the data would be using other principles, such as BPE or WordPiece.

Thirdly, as was shown in the fully upper-cased or fully de-diacritized scenarios, the best performance in such cases is reached by the systems that use flags for as long sequences as possible, for the spans of the words or even for the full lines. Thus, in future, we should try introducing such flags for other types of noising and see if their added value is comparable to the flags that are already used for other long sequences.

Finally, it would be interesting to see how the MT system reacts to the particular flags directly, for instance, by examining the attention weights assigned to the flags and their connections with other tokens.

7.2 Ethical Statement

One of our work's focuses was the improvement of the robustness of the systems against the noise. The robustness improvement can be seen as a dual use technology, if a user intentionally tries to prevent the automatic analysis of their texts. In many cases, such an activity of intentional noising can be used for illegal activities such as phishing or other type of fraud. However, in countries with oppressive political regimes, the total scrapping of the content generated by the users can be used for censorship and tracking of dissidents. Based on the experience of the author of the thesis, we hope that the scope of the noising scenarios examined here is different from the one generally used to hide oppositional content. Still, we urge the community to bear the possibility of the robust systems they develop for evil purposes.

Bibliography

Noëmi Aepli and Rico Sennrich. Improving Zero-Shot Cross-lingual Transfer Between Closely Related Languages by Injecting Character-Level Noise. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 4074–4083, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.321. URL `https://aclanthology.org/2022.findings-acl.321`.

Gustavo Aguilar, Bryan McCann, Tong Niu, Nazneen Rajani, Nitish Shirish Keskar, and Thamar Solorio. Char2Subword: Extending the Subword Embedding Space Using Robust Character Compositionality. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1640–1651, Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.141. URL `https://aclanthology.org/2021.findings-emnlp.141`.

Jesujoba Alabi, Kwabena Amponsah-Kaakyire, David Adelani, and Cristina España-Bonet. Massive vs. Curated Embeddings for Low-Resourced Languages: the Case of Yorùbá and Twi. pages 2754–2762, Marseille, France, 2020. European Language Resources Association.

Chantal Amrhein and Rico Sennrich. On Romanization for Model Transfer Between Scripts in Neural Machine Translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2461–2469, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.223. URL `https://www.aclweb.org/anthology/2020.findings-emnlp.223`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. 2014. doi: 10.48550/ARXIV.1409.0473. URL `https://arxiv.org/abs/1409.0473`. Publisher: arXiv Version Number: 7.

Jiří Balhar. Improving Subword Tokenization Methods for Multilingual Models. Master's thesis, Charles University, Prague, Czech Republic, 2023.

Satanjeev Banerjee and Alon Lavie. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. pages 65–72, Ann Arbor, Michigan, 2005. Association for Computational Linguistics. URL `https://aclanthology.org/W05-0909/`.

Lisa Beinborn and Yuval Pinter. Analyzing Cognitive Plausibility of Subword Tokenization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4478–4486, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.272. URL `https://aclanthology.org/2023.emnlp-main.272`.

Alexandre Berard, Ioan Calapodescu, and Claude Roux. Naver Labs Europe's Systems for the WMT19 Machine Translation Robustness Task. In

Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1), pages 526–532, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5361. URL `https://www.aclweb.org/anthology/W19-5361`.

Verena Blaschke, Hinrich Schütze, and Barbara Plank. Does Manipulating Tokenization Aid Cross-Lingual Transfer? A Study on POS Tagging for Non-Standardized Languages. In *Tenth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2023)*, pages 40–54, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.vardial-1.5. URL `https://aclanthology.org/2023.vardial-1.5`.

Ondřej Bojar, Evgeny Matusov, and Hermann Ney. Czech-English Phrase-Based Machine Translation. pages 214–224, Turku, Finland, 2006.

Zewen Chi, Li Dong, Furu Wei, Nan Yang, Saksham Singhal, Wenhui Wang, Xia Song, Xian-Ling Mao, Heyan Huang, and Ming Zhou. InfoXLM: An Information-Theoretic Framework for Cross-Lingual Language Model Pre-Training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3576–3588, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.280. URL `https://aclanthology.org/2021.naacl-main.280`.

Hyung Won Chung, Dan Garrette, Kiat Chuan Tan, and Jason Riesa. Improving Multilingual Models with Language-Clustered Vocabularies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4536–4546, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.367. URL `https://www.aclweb.org/anthology/2020.emnlp-main.367`.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. A Character-level Decoder without Explicit Segmentation for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1160. URL `http://aclweb.org/anthology/P16-1160`.

Marco Cognetta, Sangwhan Moon, Lawrence Wolf-sonkin, and Naoaki Okazaki. Parameter-Efficient Korean Character-Level Language Modeling. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2350–2356, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.eacl-main.172. URL `https://aclanthology.org/2023.eacl-main.172`.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised Cross-lingual Representation Learning at Scale. 2019. doi: 10.48550/ARXIV.1911.02116. URL `https://arxiv.org/abs/1911.02116`. Publisher: [object Object] Version Number: 2.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.747. URL `https://www.aclweb.org/anthology/2020.acl-main.747`.

Mathias Creutz and Krista Lagus. Unsupervised Discovery of Morphemes. 2002. doi: 10.48550/ARXIV.CS/0205057. URL `https://arxiv.org/abs/cs/0205057`. Publisher: arXiv Version Number: 1.

Marie-Catherine De Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. Universal Dependencies. *Computational Linguistics*, pages 1–54, May 2021. ISSN 0891-2017, 1530-9312. doi: 10.1162/coli_a_00402. URL `https://direct.mit.edu/coli/article/doi/10.1162/coli_a_00402/98516/Universal-Dependencies`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `http://aclweb.org/anthology/N19-1423`.

Lukas Edman, Gabriele Sarti, Antonio Toral, Gertjan van Noord, and Arianna Bisazza. Are Character-level Translations Worth the Wait? Comparing Character- and Subword-level Models for Machine Translation. 2023. doi: 10.48550/ARXIV.2302.14220. URL `https://arxiv.org/abs/2302.14220`. Publisher: arXiv Version Number: 2.

Ramy Eskander, Francesca Callejas, Elizabeth Nichols, Judith Klavans, and Smaranda Muresan. MorphAGram, Evaluation and Framework for Unsupervised Morphological Segmentation. pages 7112–7122, Marceille, France, 2020. European Language Resources Association.

Thierry Etchegoyhen and Harritxu Gete. To Case or not to case: Evaluating Casing Methods for Neural Machine Translation. pages 3752–3760, Marseille, France, 2020. European Language Resources Association. URL `https://aclanthology.org/2020.lrec-1.463/`.

Markus Freitag, George Foster, David Grangier, Viresh Ratnakar, Qijun Tan, and Wolfgang Macherey. Experts, Errors, and Context: A Large-Scale Study of Human Evaluation for Machine Translation. *Transactions of the Association for Computational Linguistics*, 9:1460–1474, December 2021. ISSN 2307-387X. doi: 10.1162/tacl_a_00437. URL `https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00437/108866/Experts-Errors-and-Context-A-Large-Scale-Study-of`.

Markus Freitag, Ricardo Rei, Mathur, Chi-kiu Lo, Craig Stewart, Eleftherios Avramidis, Tom Kocmi, George Foster, Alon Lavie, and André F. T. Martins. Results of WMT22 Metrics Shared Task: Stop Using BLEU – Neural Metrics

Are Better and More Robust. pages 46–68, Abu Dhabi, United Arab Emirates (Hybrid), 2022. Association for Computational Linguistics.

Philip Gage. A new Algorithm for Data Compression. *The C Users Journal archive*, 12:23–38, 1994. URL `https://www.semanticscholar.org/paper/A-new-algorithm-for-data-compression-Gage/1aa9c0045f1fe8c79cce03c7c14ef4b4643a21f8`.

Andargachew Mekonnen Gezmu and Andreas Nürnberger. Morpheme-Based Neural Machine Translation Models for Low-Resource Fusion Languages. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 22 (9):1–19, September 2023. ISSN 2375-4699, 2375-4702. doi: 10.1145/3610773. URL `https://dl.acm.org/doi/10.1145/3610773`.

Thamme Gowda and Jonathan May. Finding the Optimal Vocabulary Size for Neural Machine Translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.352. URL `https://www.aclweb.org/anthology/2020.findings-emnlp.352`.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. The Flores-101 Evaluation Benchmark for Low-Resource and Multilingual Machine Translation. *Transactions of the Association for Computational Linguistics*, 10:522–538, May 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00474. URL `https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00474/110993/The-Flores-101-Evaluation-Benchmark-for-Low`.

Yvette Graham, Timothy Baldwin, Alistair Moffat, and Justin Zobel. Continuous Measurement Scales in Human Evaluation of Machine Translation. pages 33–41, Sofia, Bulgaria, 2013. Association for Computational Linguistics. URL `https://aclanthology.org/W13-2305/`.

Rohit Gupta, Laurent Besacier, Marc Dymetman, and Matthias Gallé. Character-based NMT with Transformer. 2019. doi: 10.48550/ARXIV.1911.04997. URL `https://arxiv.org/abs/1911.04997`. Publisher: arXiv Version Number: 1.

Martin Haspelmath and Andrea D. Sims. *Understanding morphology*. Understanding language series. Hodder Education, London, 2nd ed edition, 2010. ISBN 978-0-340-95001-2.

Rohit Jain, Huda Khayrallah, Roman Grundkiewicz, and Marcin Junczys-Dowmunt. Perplexity-Driven Case Encoding Needs Augmentation for CAPITALIZATION Robustness. pages 146–156, Nusa Dua, Bali, 2023. Association for Computational Linguistics. URL `https://aclanthology.org/2023.ijcnlp-short.17`.

Zhiying Jiang, Matthew Yang, Mikhail Tsirlin, Raphael Tang, Yiqin Dai, and Jimmy Lin. "Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors. In *Findings of the Association for Computa-*

tional Linguistics: ACL 2023, pages 6810–6828, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.426. URL `https://aclanthology.org/2023.findings-acl.426`.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast Neural Machine Translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-4020. URL `http://aclweb.org/anthology/P18-4020`.

D. Jurafsky and J. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* 3rd edition draft edition, 2023. URL `https://web.stanford.edu/~jurafsky/slp3/`.

Vani Kanjirangat, Tanja Samardžić, Ljiljana Dolamic, and Fabio Rinaldi. Optimizing the Size of Subword Vocabularies in Dialect Classification. In *Tenth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2023)*, pages 14–30, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.vardial-1.2. URL `https://aclanthology.org/2023.vardial-1.2`.

Fabio Kepler, Jonay Trénous, Marcos Treviso, Miguel Vera, and André F. T. Martins. OpenKiwi: An Open Source Framework for Quality Estimation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 117–122, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-3020. URL `https://www.aclweb.org/anthology/P19-3020`.

Saleh Abduh Naji Ali Khoshafah and Ibraheem N.A. Tagaddeen. Effect of Diacritics on Machine Translation Performance: A Case Study of Yemeni Literature. *International Journal of Language and Literary Studies*, 5(2):324–342, July 2023. ISSN 2704-7156, 2704-5528. doi: 10.36892/ijlls.v5i2.1342. URL `https://ijlls.org/index.php/ijlls/article/view/1342`.

Margaret King. Evaluating natural language processing systems. *Communications of the ACM*, 39(1):73–79, January 1996. ISSN 0001-0782, 1557-7317. doi: 10.1145/234173.234208. URL `https://dl.acm.org/doi/10.1145/234173.234208`.

P. Koehn. Statistical Significance Tests for Machine Translation Evaluation. pages 388–395, Barcelona, Spain, 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-3250/`.

P. Koehn, H. Hoang, Alexandra Birch, M. Federico, C. Callison-Birch, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open Source Toolkit for Statistical Machine Translation. pages 177–180, Prague, Czech Republic, 2007. Association for Computational Linguistics. URL `https://aclanthology.org/P07-2045`.

Philipp Koehn. *Statistical machine translation.* Cambridge University Press, Cambridge ; New York, 2010. ISBN 978-0-521-87415-1. OCLC: ocn316824008.

Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL `http://aclweb.org/anthology/P18-1007`.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL `http://aclweb.org/anthology/D18-2012`.

Amit Kumar, Shantipriya Parida, Ajay Pratap, and Anil Kumar Singh. Machine translation by projecting text into the same phonetic-orthographic space using a common encoding. *Sādhanā*, 48(4):238, November 2023. ISSN 0973-7677. doi: 10.1007/s12046-023-02275-0. URL `https://link.springer.com/10.1007/s12046-023-02275-0`.

V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, February 1966. URL `https://ui.adsabs.harvard.edu/abs/1966SPhD...10..707L`. ADS Bibcode: 1966SPhD...10..707L.

Davis Liang, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. XLM-V: Overcoming the Vocabulary Bottleneck in Multilingual Masked Language Models. 2023. doi: 10.48550/ARXIV.2301.10472. URL `https://arxiv.org/abs/2301.10472`. Publisher: arXiv Version Number: 2.

Jindřich Libovický, Helmut Schmid, and Alexander Fraser. Why don't people use character-level machine translation? In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2470–2485, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.194. URL `https://aclanthology.org/2022.findings-acl.194`.

Tomasz Limisiewicz, Jiří Balhar, and David Mareček. Tokenization Impacts Multilingual Language Modeling: Assessing Vocabulary Allocation and Overlap Across Languages. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5661–5681, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.350. URL `https://aclanthology.org/2023.findings-acl.350`.

Nikola Ljubešić, Tomaž Erjavec, and Darja Fišer. Corpus-Based Diacritic Restoration for South Slavic Languages. pages 3612–3616, Portorož, Slovenia, 2016. European Language Resources Association (ELRA). URL `https://aclanthology.org/L16-1573/`.

Chi-kiu Lo. YiSi - a Unified Semantic MT Quality Evaluation and Estimation Metric for Languages with Different Levels of Available Resources. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 507–513, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5358. URL `https://www.aclweb.org/anthology/W19-5358`.

Arle Lommel, Hans Uszkoreit, and Aljoscha Burchardt. Multidimensional Quality Metrics (MQM): A Framework for Declaring and Describing Translation Quality Metrics. *Tradumàtica tecnologies de la traducció*, (12):455–463, December 2014. ISSN 1578-7559. doi: 10.5565/rev/tradumatica.77. URL `https://revistes.uab.cat/tradumatica/article/view/n12-lommel-uzskoreit-burchardt`.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL `http://aclweb.org/anthology/D15-1166`.

Qingsong Ma, Johnny Wei, Ondřej Bojar, and Yvette Graham. Results of the WMT19 Metrics Shared Task: Segment-Level and Strong MT Systems Pose Big Challenges. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 62–90, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5302. URL `https://www.aclweb.org/anthology/W19-5302`.

Dominik Macháček, Jonáš Vidra, and Ondřej Bojar. Morphological and Language-Agnostic Word Segmentation for NMT. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue*, volume 11107, pages 277–284. Springer International Publishing, Cham, 2018. ISBN 978-3-030-00793-5 978-3-030-00794-2. doi: 10.1007/978-3-030-00794-2_30. URL `http://link.springer.com/10.1007/978-3-030-00794-2_30`. Series Title: Lecture Notes in Computer Science.

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. Glyce: Glyph-vectors for Chinese Character Representations. 2019. doi: 10.48550/ARXIV.1901.10125. URL `https://arxiv.org/abs/1901.10125`. Publisher: arXiv Version Number: 5.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. 2021. doi: 10.48550/ARXIV.2112.10508. URL `https://arxiv.org/abs/2112.10508`. Publisher: arXiv Version Number: 1.

Nikita Moghe, Tom Sherborne, Mark Steedman, and Alexandra Birch. Extrinsic Evaluation of Machine Translation Metrics. In *Proceedings of the*

61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13060–13078, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.730. URL https://aclanthology.org/2023.acl-long.730.

Ibraheem Muhammad Moosa, Mahmud Elahi Akhter, and Ashfia Binte Habib. Does Transliteration Help Multilingual Language Modeling? In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 670–685, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-eacl.50. URL https://aclanthology.org/2023.findings-eacl.50.

Cao Hong Nga, Nguyen Khai Thinh, Pao-Chi Chang, and Jia-Ching Wang. Deep Learning Based Vietnamese Diacritics Restoration. In *2019 IEEE International Symposium on Multimedia (ISM)*, pages 331–3313, San Diego, CA, USA, December 2019. IEEE. ISBN 978-1-72815-606-4. doi: 10.1109/ISM46123.2019.00074. URL https://ieeexplore.ieee.org/document/8958999/.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural Discrete Representation Learning. *Advances in Neural Information Processing Systems,*, 30, 2017. doi: 10.48550/ARXIV.1711.00937. URL https://arxiv.org/abs/1711.00937. Publisher: arXiv Version Number: 2.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Philadelphia, Pennsylvania, 2001. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL http://portal.acm.org/citation.cfm?doid=1073083.1073135.

Martin Popel. CUNI English-Czech and English-Polish Systems in WMT20: Robust Document-Level Training. pages 269–273, Online, 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.wmt-1.28/.

Martin Popel, Marketa Tomkova, Jakub Tomek, Łukasz Kaiser, Jakob Uszkoreit, Ondřej Bojar, and Zdeněk Žabokrtský. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 11(1):4381, September 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-18073-9. URL https://www.nature.com/articles/s41467-020-18073-9.

Martin Popel, Jindřich Libovický, and Jindřich Helcl. CUNI Systems for the WMT 22 Czech-Ukrainian Translation Task. pages 352–357, Abu Dhabi, United Arab Emirates (Hybrid), 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.wmt-1.30/.

Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL http://aclweb.org/anthology/W15-3049.

Matt Post. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6319. URL `http://aclweb.org/anthology/W18-6319`.

Rafal Powalski and Tomasz Stanislawek. UniCase – Rethinking Casing in Language Models. 2020. doi: 10.48550/ARXIV.2010.11936. URL `https://arxiv.org/abs/2010.11936`. Publisher: arXiv Version Number: 1.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-Dropout: Simple and Effective Subword Regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.170. URL `https://www.aclweb.org/anthology/2020.acl-main.170`.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. page 12, 2018. URL `https://www.mikecaptain.com/resources/pdf/GPT-1.pdf`.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. COMET: A Neural Framework for MT Evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.213. URL `https://www.aclweb.org/anthology/2020.emnlp-main.213`.

Ricardo Rei, Jose G.C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. COMET-22: Unbabel-IST 2022 Submission for the Metrics Shared Task. pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid), 2022. Association for Computational Linguistics. URL `https://aclanthology.org/2022.wmt-1.52/`.

S. J. Rexline and L. Robert. Substitution coder - A reversible data transform for lossless text compression. In *2011 8th International Conference on Information, Communications & Signal Processing*, pages 1–5, Singapore, December 2011. IEEE. ISBN 978-1-4577-0031-6 978-1-4577-0029-3 978-1-4577-0030-9. doi: 10.1109/ICICS.2011.6173125. URL `http://ieeexplore.ieee.org/document/6173125/`.

Mariana Romanyshyn, editor. *Proceedings of the Second Ukrainian Natural Language Processing Workshop (UNLP)*. Association for Computational Linguistics, Dubrovnik, Croatia, 2023. URL `https://aclanthology.org/2023.unlp-1.0`.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.243. URL https://aclanthology.org/2021.acl-long.243.

David Samuel and Lilja Øvrelid. Tokenization with Factorized Subword Encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 14143–14161, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.890. URL https://aclanthology.org/2023.findings-acl.890.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, Boston, MA, USA, June 2015. IEEE. ISBN 978-1-4673-6964-0. doi: 10.1109/CVPR.2015.7298682. URL http://ieeexplore.ieee.org/document/7298682/.

Mike Schuster and Kaisuke Nakajima. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, Kyoto, Japan, March 2012. IEEE. ISBN 978-1-4673-0046-9 978-1-4673-0045-2 978-1-4673-0044-5. doi: 10.1109/ICASSP.2012.6289079. URL http://ieeexplore.ieee.org/document/6289079/.

Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. WikiMatrix: Mining 135M Parallel Sentences in 1620 Language Pairs from Wikipedia. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1351–1361, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.115. URL https://aclanthology.org/2021.eacl-main.115.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.704. URL https://www.aclweb.org/anthology/2020.acl-main.704.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL http://aclweb.org/anthology/P16-1162.

Hossam Shamardan and Yasser Hifny. Arabic Diacritics Restoration Using Maximum Entropy Language Models. *IEEE Signal Processing Letters*, 30:1227–1231, 2023. ISSN 1070-9908, 1558-2361. doi: 10.1109/LSP.2023.3295752. URL https://ieeexplore.ieee.org/document/10184906/.

Xuewen Shi, Heyan Huang, Ping Jian, and Yi-Kun Tang. Case-Sensitive Neural Machine Translation. In Hady W. Lauw, Raymond Chi-Wing Wong, Alexandros Ntoulas, Ee-Peng Lim, See-Kiong Ng, and Sinno Jialin Pan, editors, *Advances in Knowledge Discovery and Data Mining*, volume 12084, pages 662–674. Springer International Publishing, Cham, 2020. ISBN 978-3-030-47425-6 978-3-030-47426-3. doi: 10.1007/978-3-030-47426-3_51. URL http://link.springer.com/10.1007/978-3-030-47426-3_51. Series Title: Lecture Notes in Computer Science.

Chenglei Si, Zhengyan Zhang, Yingfa Chen, Fanchao Qi, Xiaozhi Wang, Zhiyuan Liu, Yasheng Wang, Qun Liu, and Maosong Sun. Sub-Character Tokenization for Chinese Pretrained Language Models. *Transactions of the Association for Computational Linguistics*, 11:469–487, May 2023. ISSN 2307-387X. doi: 10.1162/tacl_a_00560. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00560/116047/Sub-Character-Tokenization-for-Chinese-Pretrained.

Aryan Singh and Jhalak Bansal. Neural Machine Transliteration Of Indian Languages. In *2021 4th International Conference on Computing and Communications Technologies (ICCCT)*, pages 91–96, Chennai, India, December 2021. IEEE. ISBN 978-1-66541-447-0. doi: 10.1109/ICCCT53315.2021.9711806. URL https://ieeexplore.ieee.org/document/9711806/.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. A Study of Translation Edit Rate with Targeted Human Annotation. pages 223–231, Cambridge, Massachusetts, USA, 2006. Association for Machine Translation in the Americas. URL https://aclanthology.org/2006.amta-papers.25/.

Lucia Specia, Kashif Shah, Jose G.C. de Souza, and Trevor Cohn. QuEst - A translation quality estimation framework. pages 79–84, Sofia, Bulgaria, 2013. Association for Computational Linguistics. URL https://aclanthology.org/P13-4014/.

Lucia Specia, Carolina Scarton, and Gustavo Henrique Paetzold. *Quality Estimation for Machine Translation.* Synthesis Lectures on Human Language Technologies. Springer International Publishing, Cham, 2018. ISBN 978-3-031-01040-8 978-3-031-02168-8. doi: 10.1007/978-3-031-02168-8. URL https://link.springer.com/10.1007/978-3-031-02168-8.

Aarohi Srivastava and David Chiang. Fine-Tuning BERT with Character-Level Noise for Zero-Shot Transfer to Dialects and Closely-Related Languages. In *Tenth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2023)*, pages 152–162, Dubrovnik, Croatia, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.vardial-1.16. URL https://aclanthology.org/2023.vardial-1.16.

Milos Stanojevic and Khalil Sima'an. BEER: BEtter Evaluation as Ranking. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 414–419, Baltimore, Maryland, USA, 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3354. URL http://aclweb.org/anthology/W14-3354.

Vasyl Starko, Andriy Rysin, and Maria Shvedova. Ukrainian Text Preprocessing in GRAC. In *2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT)*, pages 101–104, LVIV, Ukraine, September 2021. IEEE. ISBN 978-1-66544-257-2. doi: 10.1109/CSIT52700.2021.9648705. URL https://ieeexplore.ieee.org/document/9648705/.

Jörg Tiedemann. Parallel Data, Tools and Interfaces in OPUS. pages 2214–2218, Istanbul, Turkey, 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 2017. doi: 10.48550/ARXIV.1706.03762. URL https://arxiv.org/abs/1706.03762. Publisher: arXiv Version Number: 7.

Tereza Vojtěchová, Michal Novák, Miloš Klouček, and Ondřej Bojar. SAO WMT19 Test Suite: Machine Translation of Audit Reports. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 481–493, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5355. URL https://www.aclweb.org/anthology/W19-5355.

John S. White, Theresa O'Connell, and Francis O'Mara. The ARPA MT Evaluation Methodologies: Evolution, Lessons, and Future Approaches. pages 193–205, Columbia, Maryland, USA, 1994. URL https://aclanthology.org/1994.amta-1.25/.

Benoist Wolleb, Romain Silvestri, Giorgos Vernikos, Ljiljana Dolamic, and Andrei Popescu-Belis. Assessing the Importance of Frequency versus Compositionality for Subword-based Tokenization in NMT. 2023. doi: 10.48550/ARXIV.2306.01393. URL https://arxiv.org/abs/2306.01393. Publisher: arXiv Version Number: 2.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. 2016. doi: 10.48550/ARXIV.1609.08144. URL https://arxiv.org/abs/1609.08144. Publisher: arXiv Version Number: 2.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL https://aclanthology.org/2021.naacl-main.41.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models. *Transactions of the Association for Computational Linguistics*, 10:291–306, March 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00461. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00461/110049/ByT5-Towards-a-Token-Free-Future-with-Pre-trained.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT, February 2020. URL http://arxiv.org/abs/1904.09675. arXiv:1904.09675 [cs].

Bo Zheng, Li Dong, Shaohan Huang, Saksham Singhal, Wanxiang Che, Ting Liu, Xia Song, and Furu Wei. Allocating Large Vocabulary Capacity for Cross-Lingual Language Model Pre-Training. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3203–3215, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.257. URL https://aclanthology.org/2021.emnlp-main.257.

Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. Tokenization and the Noiseless Channel. 2023. doi: 10.48550/ARXIV.2306.16842. URL https://arxiv.org/abs/2306.16842. Publisher: arXiv Version Number: 1.

List of Figures

2.1 Average number of diacritized characters for each word length, calculated on the T8M training dataset (only Czech side). The x-axis represents the word length in characters (only subset of 1 to 25 characters is taken since it covers the majority of the words). The left y-axis shows the average number of diacritized characters for each word length; the two lines are showing the values (red is computed on unique word types, blue is weighted by the frequencies of each word type). The right y-axis shows the share of the cumulative distribution function of the words coverage in the dataset; the grey area is showing its values. 31

2.2 Average Levenshtein distance of the diacritization variants (ranked by frequency) for the Czech data. The x-axis represents the Word-InDia diacritization flags in ascending order. The number represents the frequency rank. On the right table, the "pivotal" words from which distance is computed is the base, thus all ranks (including most the most frequent, denoted by "0" flag) are shown. On the left table, the most frequent version is not presented as it is pivotal point itself; the negative "X" flag represents the base form in case it is present in the training data and different from the most frequent one. The y-axis represents the Levenshtein distance between each rank and the "pivot", which is averaged over the whole InDia vocabulary entries. 32

4.1 Comparison of the Rényi efficiency score depending on alpha. The subplots are created for each type of case noising, each figure shows the EFF score of each system (Y axis) with respect to alpha score (X axis). 71

4.2 Counts of the number of unique words that were met in the corpus a particular number of times. The upper-cased and the title-cased words are marked separately. The x-axis represents the number of times a particular unique word was met in the training data. The y-axis represents how much words were met a given number of times. Both axes are logarithmic. 79

4.3 Comparison of the intrinsic metrics for the tokenizers trained on InCa vocabularies with different minimal counts. The first row shows the character per token ratio; the second one – the Average Rank ratio (defined in 2.4). The left column shows the performance on the Czech data, the right – on the Ukrainian data. Each graph shows the efficiency of the metric (y-axis) with respect to minimal counts value (x-axis, logarithmic). On each graph, four lines represent different noising of the encoded dataset: black for no noising ("none"), red for fully upper-cased; green for fully lower-cased; blue for random 10% casing. 80

7.1 Pairwise comparison of the main extrinsic and intrinsic metrics on all experiments. In each graph, the x-axis represents the intrinsic metric score and the y-axis – the extrinsic metric score. The colours represent the language of the encoded text (and, for extrinsic metric, the source) – red is for Czech (and Czech-Ukrainian direction), blue is for Ukrainian (and Ukrainian-Czech direction). The ρ stands for Pearson correlation coefficient, the r_s stands for Spearman's rank correlation coefficient; the asterisk shows if the p-value is lower than 0.05. 100
7.2 Comparison of two intrinsic metric distributions – character per token (CPT, x-axis) and average rank (AR, y-axis). The colours represent the language of the encoded text – red is for Czech, blue is for Ukrainian. The ρ stands for Pearson correlation coefficient, the r_s stands for Spearman's rank correlation coefficient; the asterisk shows if the p-value is lower than 0.05. 101
7.3 Comparison of two intrinsic CPT_v score on the tokenizer and the intrinsic (CPT) and extrinsic (BLEU) scores for the texts generated by the tokenizers. The CPT_v score is shown over x-axis, the extrinsic and intrinsic metrics are shown over y-axis. Each data point shows a metric for a particular text; they differ by the color depending on the type of noising of the text which is encoded. . 102

List of Tables

1 An illustration of the tokenization problem with the phrase (in the second column) which was transformed either by upper-casing or by deleting the diacritization. For a general Czech speaker, all three phrases would be easily understood as the same phrase; however, the tokeniser struggles to split it consistently as it was trained on the "regular" data, where the words are diacritized and usually not upper-cased. In the third column, you see how the tokenizer splits the sequence into subwords (the splitting is represented by white spaces, the underscore is an auxiliary tokenizer sign denoting the actual white space). Contrary to the "normal" sentence in the first line, where the line is split into meaningful Czech words (or at least morphemes), in two lower lines it is over-tokenized into meaningless parts, which would influence the processing time and the performance on the downstream NLP task. 4

2.1 Dataset Overview. The number of sentences and words was counted as the number of lines and the number of white space-delimited positive-length character sequences, respectively. 22

2.2 Data preparation. The symbols mentioned in the "ID" column will be used to denote the text modifications in the ablation studies. The "Type" column shows which aspect of noising it refers to (either casing or diacritization); the "Language" column refers to the languages to which the noising applies; the "Aligned" column shows whether the operation is applied simultaneously to the pair of sentences or its application on the Czech and the Ukrainian sentence does not have to be aligned. 24

2.3 Overview of the pre-processing algorithms. The names in the "ID" column and the parameter labels in the "Params and Variants" column will be consistently used hereinafter to refer to the experiment configurations. The "Type" column describes whether this preprocessing is applied for the casing or diacritization experiments. 35

2.4 Overview of the metrics used in the analysis. The names in the "ID" column will be consistently used to denote the metrics hereinafter. The "Level" column shows whether the metrics perform the extrinsic (`ext`) or intrinsic (`int`) evaluation. The column "Range" shows the range of the possible values of the metrics ($|V|$ stands for the tokenizer vocabulary size). For all metrics except for flag ranks the higher value means the better performance. 48

3.1 Table 3.1. Counts of occurrences of various types of apostrophes in the Ukrainian parts of the training (T8M) and test (flo) datasets 52

3.2 Internal and external metrics comparison with respect to apostrophe normalization, Czech-Ukrainian translation direction. The first two columns show the training parameters of the MT model, each line corresponds to a separately trained system. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Normalization" shows whether we used no mapping to the only apostrophe (-), mapping to the ASCII apostrophe (ASCII), mapping to ASCII apostrophe with disabled SentencePiece parameter "split_by_unicode_script" (ASCII, -split). The metric naming is defined in 2.4. 53
3.3 Internal and external metrics comparison with respect to apostrophe normalization, Ukrainian-Czech translation direction. The notation is equivalent to the one in Table 3.2 53
3.4 SentencePiece vocabulary statistics on apostrophe normalization modes. The first two columns show the training parameters of the SentencePiece model, each line corresponds to a separately trained tokenizer. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Normalization" shows whether we used no mapping to the only apostrophe (-), mapping to the ASCII apostrophe (ASCII), mapping to ASCII apostrophe with disabled SentencePiece parameter "split_by_unicode_script" (ASCII, -split) or mapping to the Ukrainian modifier letter apostrophe with the disabled "split_by_unicode_script" parameter (letter, -split). The "Apostrophe" column denotes which apostrophe character we would count in the "Tokens with Apostrophe" column – ASCII one (ASCII) or the "Modifier letter apostrophe" (letter). "CPT$_v$" denotes the average token length in the vocabulary, "Tokens with Apostrophe" show the number of the tokens that contain the apostrophe of a given type. 55
3.5 External metrics comparison with respect to upper-case paired sentence normalization, Czech-Ukrainian translation direction. The first two columns show the training parameters of the MT model, each line corresponds to a separately trained system. The "InCa" column shows preprocessing setup (- for no preprocessing, + for standard InCa); "Norm. Upper" shows whether we used uppercase preprocessing for the consistency of source and target sentence pairs (- for non-normalized data, + for normalized data). The models are evaluated on the fully upper-cased flo dataset and on the default flo dataset (described in Section 2.1); since the main interest is fully upper-cased dataset, it is provided in general font, while the subscripts show the performance on the default dataset for reference. The metrics are decscribed in Section 2.4.1. 57
3.6 External metrics comparison with respect to upper-case paired sentence normalization, Ukrainian-Czech translation direction. The notation is equivalent to the one described in Table 3.5. 57

3.7 Extrinsic metrics comparison of the MT systems trained in the same setup: no preprocessing, Czech-Ukrainian direction. Each line represents an instance of the model trained separately but with the same setup and hyper-parameters. The BLEU and chrF columns comprise three parameters: “Score” is an overall score computed for the whole document; “$\mu \pm 95\% CI$” is mean score and 95% confidence interval based on bootstrap resampling of the given system output; “p-value” is significance test comparing the first run with the given run (significance threshold is 0.05, statistically significant values are marked with an asterisk). The COMET value shows the score only. 58
3.8 Extrinsic metrics comparison of the MT systems trained in the same setup: standard InCa preprocessing, Czech-Ukrainian direction. The notation is equivalent to one in 3.7. 58
3.9 Extrinsic metrics comparison of the MT systems trained in the same setup: no preprocessing, Ukrainian-Czech direction. The notation is equivalent to one in 3.7. 59
3.10 Extrinsic metrics comparison of the MT systems trained in the same setup: standard InCa preprocessing, Ukrainian-Czech direction. The notation is equivalent to one in 3.7. 59
3.11 Extrinsic metrics comparison of the combined outputs of 3 runs of MT systems trained in the same setup, Czech-Ukrainian direction. The “preprocessing” column shows whether we used InCa (“inca”) or no pre-processing (“base”). The BLEU and chrF columns comprise three parameters: “Score” is an overall score computed for the whole document; “$\mu \pm 95\% CI$” is mean score and 95% confidence interval based on bootstrap resampling of the given system output; “p-value” is significance test comparing the first and the second lines (significance threshold is 0.05). 60
3.12 Extrinsic metrics comparison of the combined outputs of 3 runs of MT systems trained in the same setup, Ukrainian-Czech direction. The notation is equivalent to one in 3.11. 60

4.1 Illustration on the work of different preprocessing algorithms (and tokenizers applied to the pre-processed texts) with respect to casing. The first row shows the examples of the input sentence; the next rows show the results of the tokenization (the boundary between tokens is shown with a white space). The flags that were used at the pre-processing step are marked blue. The flag T means title-casing of the next word, the flag U upper-cases the whole word. There are other flags that are not represented in the example (for instance, fully upper-cased line or upper-cased spans) that will be shown later. For TokenMonster preprocessing, the square brackets show that the multi-word token boundaries. The preprocessing algorithms’ naming follows 2.3. 62

4.2 Extrinsic metrics comparison of the main inline casing algorithms for the general (not noised) `flo` development set, Czech-Ukrainian translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 62
4.3 Extrinsic metrics comparison of the main inline casing algorithms for the general (not noised) `flo` development set, Ukrainian-Czech translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 63
4.4 Extrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Czech-Ukrainian translation pair. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 63
4.5 Extrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Ukrainian-Czech translation pair. The legend follows the corresponding Czech-Ukrainian table above. 64
4.6 Examples of flag hallucinations that took place in the Ukrainian-Czech translations of the fully upper-cased texts. Each preprocessor (left column) shows the flags that are used to represent upper-case and other casing operations (central column). For each preprocessor, the rightmost column shows at the first line the tokenized output generated by the MT system and then the final output after the post-processing algorithm. 65
4.7 Intrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Czech data. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 66
4.8 Intrinsic metrics comparison of the main inline casing algorithms for the fully upper-cased noising of the `flo` dataset, Ukrainian data. The legend follows the table above. 66
4.9 Average Rank and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Czech texts. The names of noising modes follow 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4. The flags are denoted as follow: "T" stands for title-case, "U" – for upper-casing a word, "A" – for upper-casing the whole sentence (or a span for marian), "-A" – for ending the upper-cased span for marian, "L" – for lower-casing the word. The bold figures mark either the best values over the column; for AR scores higher is better, for "Variation" – lower is better. 67
4.10 Average Rank and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Ukrainian texts. The legend conventions follow the table 4.9 on Czech data. 68

4.11 Rényi efficiency and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Czech texts. The names of noising modes follow 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4. The flags are denoted as follow: "T" stands for title-case, "U" – for upper-casing a word, "A" – for upper-casing the whole sentence (or a span for marian), "-A" – for ending the upper-cased span for marian, "L" – for lower-casing the word. The best (highest) scores for each column are marked bold. 69
4.12 Rényi efficiency and ranks of the casing flags for various types of noising in the `flo` dataset, encoded Ukrainian texts. The legend conventions follow the table on Czech data above. 70
4.13 Vocabulary statistics for the tokenizers trained for the main models. The CPT_v metric is defined in 2.4; the "Cased tokens" value shows the number of tokens that contain a casing flag (or cased letters), and the "Overlap with Uncased" value shows the number of the uncased tokens in the dictionary that differ from the cased ones (in the "Cased tokens") only by casing or casing flag. 72
4.14 Extrinsic performance in fully upper-cased scenario, Czech-Ukrainian translation direction. The metrics are formulated as in 2.4. The standard InCa implementation is referred to as "inca" in the "Preprocessing" column, and the ablation without special treatment of the full upper-case sentence is referred to as "inca-A". 73
4.15 Extrinsic performance in fully upper-cased scenario, Ukrainian-Czech translation direction. The legend conventions follow the table on Czech-Ukrainian table above. 73
4.16 Intrinsic performance in fully upper-cased scenario, encoded Czech text. The metrics are formulated as in 2.4. The standard InCa implementation is referred to as "inca" in the "Preprocessing" column, and the ablation without special treatment of the full upper-case sentence is referred to as "inca-A". 73
4.17 Intrinsic performance in fully upper-cased scenario, encoded Ukrainian text. The legend conventions follow the table on Czech table above. 74
4.18 Extrinsic performance on casing-augmented data, Czech-Ukrainian translation direction. The metrics are formulated as in 2.4, they show scores for fully lower-cased noising an fully upper-case noising of the development dataset. In the "Setup" column, the non-preprocessing scenario and standard InCa implementation are referred to as "base" and "inca", while "base+caseaug" means casing augmentation for no-preprocessing system, "base+caseraug$_{all}$" stands for casing augmentation applied both for InCa vocabulary and for tokenizer; and "inca+caseaug$_{MT}$" stands for applying casing augmentation applied only on the tokenization and MT step. The horisontal line shows the border between the cluster of the best-scoring systems (differing from each other for less than 1 BLEU point): the rows below are the best cluster and the rows above are considerably worse. 76

4.19 Extrinsic performance on casing-augmented data, Ukrainian-Czech translation direction. The legend conventions follow the table on Czech-Ukrainian table above. 76
4.20 Intrinsic metrics on casing-augmented data, Czech text. The metrics are formulated as in 2.4, they show scores for fully lower-cased noising an fully upper-case noising of the development dataset. The values of the "Setup" column are equal to the ones defined in 4.18. 76
4.21 Intrinsic metrics on casing-augmented data, Ukrainian text. The legend conventions follow the Czech table above. 77
4.22 Average unique token length in the tokenizers with and without casing augmentation. 77

5.1 Illustration of two inline diacritization methods applied to the Czech excerpts. Two first rows show the inputs, the next lines show the results of pre-processing and tokenization ("Base" means no pre-processing). The InDia flags are marked in blue. For char-InDia, KV flag marks the separator between the keys (indices of the diacritized character) and values (flags for each character diacritization), and ID is a separator if there are multiple keys. k means putting "kroužek" diacritization, č means putting "čárka" diacritization, n means de-diacritizing the letter. For word-InDia, N means de-diacritizing the whole word, 1 and 2 mean the second- and the third- frequent diacritizations for the same base. The pre-processing algorithms' naming follows 2.3. 83
5.2 Extrinsic metrics comparison of the main inline diacritization algorithms for the general (not noised) `flo` development set, Czech-Ukrainian and Ukrainian-Czech translation pairs. The preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 84
5.3 Extrinsic metrics comparison of the main inline diacritization algorithms for the noised `flo` development set, Czech-Ukrainian translation direction. The noising naming follows 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 84
5.4 Intrinsic metrics comparison of the main inline diacritization algorithms for the un-noised and noised `flo` development set, Czech texts. The noising naming follows 2.2, the preprocessing algorithms' naming follows 2.3, the metrics are formulated as in 2.4 . 85
5.5 An example of the fully de-diacritized text tokenized by no-preprocessing (base) and two InDia algorithms. At the first two rows, the initial (diacritized) and the de-diacritized input are shown. In the InDia outputs, the blue characters represent the flags, out of which the N means stripping diactirization (for Word-InDia – full word, for Char-InDia – for particular character). For Char-InDia, KV shows the separation between the key-value parts of the flag, and the ID is a delimiter of the indices of the characters that should be diacritized in the word. 86

5.6 Average unique token length in the SentencePiece vocabularies with and without inline diacritization. The metric CPT$_v$ is defined in 2.4. . . . 86
5.7 Examples of the clusters of the Char-InDia diacritization operations flags that are generalized by the SentencePiece tokenizer and seen in the encoded flores dataset. The h flag means applying háček, č flag means čárka, n flag means de-diacritization of a character. The flags KV and ID are auxiliary tokens that delimiter the indices of the diacritized flags and their operations, the numbers in blue show the indices of the characters (0-based) to which the diacritization operations should be applied. . . . 87
5.8 Extrinsic performance on diacritization-augmented data, Czech-Ukrainian translation. The metrics are formulated as in 2.4, they show scores for fully de-diacritized and 20% de-diacritization noising of the development dataset. The “Setup” naming denotes the systems the following way: “base” and “char-InDia” stand for the no-preprocessing and Char-InDia systems trained on initial data; “+Aug” suffix means the training data was augmented with de-diacritized texts as described above. The horizontal line between the metrics shows the border between the group of the best scoring results (below the line) and the results that are at least 1 BLEU point worse (above the line). . . . 89
5.9 Intrinsic performance on diacritization-augmented data, Czech texts. We compare the fully de-diacritized and 20% de-diacritization noising of the development dataset. The system naming conventions follow the extrinsic table 5.8 above. The figures marked bold show the best results in each parameter and setup. . . . 89
5.10 Extrinsic performance comparison of standard Char-InDia and naive Char-India (marked “Char-InDia-n”), Czech-Ukrainian translation. The metrics are formulated as in 2.4, each two lines show comparison on no noising, full de-diacritization and 20% de-diacritization of the development dataset (defined in “Noise” column). . . . 91
5.11 Intrinsic performance comparison of standard Char-InDia and naive Char-India (marked “Char-InDia-n”), Czech texts. The system naming conventions follow the extrinsic table 5.10 above. . . . 91

6.1 Extrinsic performance comparison of no preprocessing and two romanization preprocessing techniques for Czech-Ukrainian and Ukrainian-Czech translation directions. The metrics are formulated as in 2.4. In column “Romanization”, “none” stands for the “base” experiments without preprocessing, “roman” stands for romanization of all characters except for the soft sign, and “roman$_{+soft}$” stands for romanization of all characters including the soft sign. . 92
6.2 Intrinsic performance comparison of no preprocessing and two romanization preprocessing techniques for Czech and Ukrainian texts. The system naming conventions follow the extrinsic table 6.1 above. . . . 93

6.3 Average unique token length in the SentencePiece tokenizer vocabularies for no preprocessing and two romanization preprocessing techniques. The system naming conventions follow the extrinsic table 6.1 above. 93
6.4 Illustration of the romanization experiments on the encoded Czech and Ukrainian sentence. The first line shows the input sentence before (possibly romanization and) tokenization. The overlapping tokens in the two languages are marked blue. The system naming conventions follow the extrinsic table 6.1 above. 94
6.5 The degree of overlap in the encoded Czech and Ukrainian bitext. The "Overlap" column stands for the count of unique tokens met in both texts (bigger is better), "IoU" stands for Intersection-over-Union score (fraction of overlap value by the number of unique tokens used in either of the texts), and "JSD" stands for Jensen-Shannon Distance (scale 0-1, lower is better). The system naming conventions follow the extrinsic table 6.1 above. 94

7.1 Spearman's rank correlation between the intrinsic metrics (denoted above) and the extrinsic metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The metrics are formulated as in 2.4. The statistically significant scores with p-value lower than 0.05 are marked with an asterisk. 97
7.2 Kendall's tau correlation between the intrinsic metrics (denoted above) and the extrinsic metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The statistically significant scores with p-value lower than 0.05 are marked with an asterisk. . 97
7.3 Spearman's rank correlation between the CPT metric (denoted above) and the standard and lower-cased versions of BLEU and chrF metrics (denoted for each column below), split by language ("cs" for Czech, "uk" for Ukranian) and noising of the flores dataset (naming follows 2.3). The metrics are formulated as in 2.4. The statistically significant scores with p-value lower than 0.05 are marked with an asterisk. 98

List of Abbreviations

A. Appendix

A.1 General Statistics on Main Inline Casing Algorithms

Noise	Preprocessing	CPT	AR	EFF	BLEU$_{(lc)}$	chrF$_{(lc)}$	COMET
none	base	3.973	1238	0.538	$21.6_{22.1}$	$51.3_{51.8}$	0.869
none	inca	3.995	1166	0.522	$21.7_{22.4}$	$51.4_{52.1}$	0.870
none	inca-n	3.592	1042	0.423	$21.9_{22.4}$	$51.4_{52.0}$	0.872
none	marian	4.033	1354	0.554	$21.9_{22.5}$	$51.7_{52.2}$	0.876
none	tkm	3.619	1101	0.492	$21.4_{21.9}$	$51.1_{51.6}$	0.870
lower	base	3.924	1047	0.539	$18.7_{20.9}$	$49.4_{50.6}$	0.849
lower	inca	3.671	1069	0.444	$19.2_{21.8}$	$50.1_{51.6}$	0.856
lower	inca-n	4.123	1193	0.527	$19.2_{21.8}$	$49.7_{51.3}$	0.855
lower	marian	4.115	1265	0.581	$18.9_{21.5}$	$49.8_{51.3}$	0.859
lower	tkm	3.760	1135	0.452	$18.7_{20.9}$	$49.4_{50.7}$	0.850
rand$_{0.1}$	base	3.745	1233	0.549	$19.9_{20.6}$	$49.4_{50.3}$	0.839
rand$_{0.1}$	inca	3.715	1085	0.473	$20.5_{22.0}$	$50.0_{51.7}$	0.855
rand$_{0.1}$	inca-n	3.394	985	0.391	$20.6_{21.8}$	$50.2_{51.4}$	0.857
rand$_{0.1}$	marian	3.907	1324	0.529	$21.0_{21.8}$	$50.9_{51.9}$	0.863
rand$_{0.1}$	tkm	3.509	1063	0.489	$20.2_{21.2}$	$50.1_{51.2}$	0.854
upper	base	1.625	60	0.658	$1.6_{1.9}$	$22.5_{23.0}$	0.448
upper	inca	3.890	1134	0.500	$21.3_{21.3}$	$51.3_{51.3}$	0.871
upper	inca-n	3.870	1121	0.488	$20.7_{20.7}$	$50.7_{50.7}$	0.867
upper	marian	3.917	1213	0.551	$15.5_{20.4}$	$39.1_{50.7}$	0.814
upper	tkm	2.434	647	0.226	$15.5_{17.9}$	$46.6_{48.9}$	0.840

Overview of the intrinsic and extrinsic metrics for the main Inline casing algorithms, Czech-Ukrainian translation direction. The "Noise" column shows which type of noising was applied (the description of the values are explained in 2.1), the "Preprocessing" table shows which case preprocessing algorithms were applied, where "base" means no preprocessing and "inca-n" means naive InCa, and all other algorithms are explained in 2.3; all metrics are defined in 2.4. The sub-scripted values under BLEU and chrF metrics show the lc(BLEU) and lc(chrF) metrics, correspondingly.

Noise	Preprocessing	CPT	AR	EFF	BLEU$_{(lc)}$	chrF$_{(lc)}$	COMET
none	base	4.033	1189	0.516	$22.7_{23.2}$	$51.0_{51.5}$	0.873
none	inca	4.014	1113	0.500	$22.7_{23.3}$	$51.0_{51.7}$	0.867
none	inca-n	3.635	997	0.417	$23.2_{23.7}$	$51.2_{51.8}$	0.873
none	marian	4.197	1301	0.563	$23.3_{23.7}$	$51.4_{51.9}$	0.875
none	tkm	4.062	1336	0.503	$22.9_{23.3}$	$51.0_{51.5}$	0.870
lower	base	4.010	1024	0.519	$19.6_{22.1}$	$49.3_{50.6}$	0.847
lower	inca	3.739	1034	0.442	$20.4_{22.9}$	$50.1_{51.5}$	0.853
lower	inca-n	4.160	1135	0.504	$19.9_{22.7}$	$49.4_{51.1}$	0.854
lower	marian	4.298	1235	0.602	$20.1_{22.8}$	$49.6_{51.1}$	0.853
lower	tkm	4.237	1383	0.474	$19.9_{22.3}$	$49.3_{50.7}$	0.846
rand$_{0.1}$	base	3.785	1154	0.527	$21.2_{21.9}$	$49.5_{50.5}$	0.844
rand$_{0.1}$	inca	3.756	1041	0.460	$21.5_{22.9}$	$49.9_{51.6}$	0.850
rand$_{0.1}$	inca-n	3.450	946	0.388	$22.0_{23.1}$	$50.3_{51.5}$	0.859
rand$_{0.1}$	marian	4.069	1266	0.534	$22.5_{23.4}$	$50.6_{51.7}$	0.862
rand$_{0.1}$	tkm	3.931	1289	0.500	$21.9_{22.8}$	$50.2_{51.2}$	0.856
upper	base	1.569	46	0.678	$1.9_{2.5}$	$21.8_{23.2}$	0.419
upper	inca	3.944	1091	0.486	$22.8_{22.8}$	$51.3_{51.3}$	0.865
upper	inca-n	3.915	1069	0.473	$22.0_{22.0}$	$50.8_{50.8}$	0.861
upper	marian	4.102	1181	0.566	$17.6_{22.3}$	$41.4_{51.0}$	0.822
upper	tkm	2.702	755	0.219	$17.6_{19.9}$	$47.4_{49.3}$	0.842

Overview of the intrinsic and extrinsic metrics for the main Inline casing algorithms, Ukrainian-Czech translation direction. The legend is the same as in A.1.