Correcting Diacritics and Typos with ByT5 Transformer Model

Lukas Stankevičius\textsuperscript{1,\dagger}, Mantas Lukoševičius\textsuperscript{1,‡}, Jurgita Kapociūtė-Dzikienė\textsuperscript{2}, Monika Briedienė\textsuperscript{2}, Tomas Krilavičius\textsuperscript{2}

\textsuperscript{1}Kaunas University of Technology
\textsuperscript{2}Vytuutas Magnus University
\dagger lukas.stankevicius@ktu.lt, \‡ mantas.lukoceivecicis@ktu.lt

January 31, 2022

Abstract

Due to the fast pace of life and online communications, the prevalence of English and the QWER TY keyboard, people tend to forgo using diacritics, make typographical errors (typos) when typing. Restoring diacritics and correcting spelling is important for proper language use and disambiguation of texts for both humans and downstream algorithms. However, both of these problems are typically addressed separately, i.e., state-of-the-art diacritics restoration methods do not tolerate other typos. In this work, we tackle both problems at once by employing newly-developed ByT5 byte-level transformer models. Our simultaneous diacritics restoration and typos correction approach demonstrates near state-of-the-art performance in 13 languages, reaching >96% of the alpha-word accuracy. We also perform diacritics restoration alone on 12 benchmark datasets with the additional one for the Lithuanian language. The experimental investigation proves that our approach is able to achieve comparable results (>98%) to previously reported despite being trained on fewer data. Our approach is also able to restore diacritics in words not seen during training with >76% accuracy. We also show the accuracies to further improve with longer training. All this shows a great real-world application potential of our suggested methods to more data, languages, and error classes.

Keywords: Natural language processing; Diacritics restoration; Typo correction; Transformer models; ByT5; QWER TY

1 Introduction

Since the dawn of the computer era, the English language, Latin alphabet, and the QWERTY keyboard are the “computer-native” means of communication. English remains the lingua franca in IT, science, and many other fields, and many people use it in addition to other, their native languages, as do we here.

Most other languages that use a Latin-based alphabet have some diacritic signs ("Æ") added to the basic Latin characters ("œ") modifying their pronunciation. The initial ASCII character set was greatly expanded by the wide adoption of the Unicode Standard to accommodate for characters of other languages. Typing these characters, however, is not always very convenient.
Even though many different keyboard layouts exist, they can be more efficient for other languages, as well as English, it is easy to remap physical keyboards in software, and virtual keyboards on touchscreens can even be dynamic, learning to type efficiently on different layouts is not easy. They are also not universally available. In addition, large alphabets are not practical to fit on a keyboard layout so that each character can be typed by pressing just one key and require combinations or sequences of keys.

All these factors made the QWERTY variations (including similar QWERTZ and AZERTY) remain the most popular keyboard layouts for Latin-alphabet-based languages, where the diacritics are usually an afterthought.

By necessity, hurry, or convenience, people often forgo the diacritic signs and special characters in these languages and type using the base Latin alphabet and keyboard layout instead. Such texts can typically be largely understood nonetheless, but introduce ambiguities and are not considered proper use of language.

Our aim in this work is to investigate automatic methods of restoring diacritic signs in such texts as well as correcting other typical typographic errors, colloquially known as “typos”, as such fast sloppy typing usually results in both.

Restoring diacritics (as well as correcting typos) is important for human readability of the texts, disambiguation, the proper use of the language (and the prestige associated with it), preventing its degradation.

On the more objectively-measurable technical side, undiacritized texts are also harder to analyze automatically: machine-translate, synthesize, parse, etc. The relevance and importance of diacritics restoration are revealed by evaluating them on the downstream tasks, i.e., extrinsically. There are several examples. The diacritics restoration helped to increase automatic speech recognition quality for the Romanian language when diacritics were restored in the corpus used for the language model training [1, 2]. Diacritics restoration also resulted in better text-to-speech performance for Romanian [3]. Used as the integrative NLU component, diacritics restoration also improved the accuracy of the intent classification-based Vietnamese dialogue system [4, 5]. Besides, statistical machine translation performance was positively correlated with correctly diacritized words for Arabic [6]. Moreover, higher binary classification accuracy was achieved after Turkish text diacritization [7].

Usually, the progress in any Natural Language Processing (NLP) topic initially begins with research for the English language and then spreads to others, but the omitted diacritics problem is an exception. The English written language is highly dependent on the original Latin alphabet. Undiacritized ASCII equivalents of a few English loanwords with diacritics (as “café”, “ naïve”, “ façade”, etc., mostly borrowed from French) do not cause ambiguity and therefore can be easily restored with a dictionary. The level of ambiguity and complexity of restoration for the other languages strongly depends on the language characteristics. For languages where omitted diacritics cause fewer disambiguation problems, the diacritics restoration is formulated as the spelling correction task. In this research, our focus is on languages that already have lexical and inflective ambiguity. Hence, the omitted diacritics exacerbate this problem even more, and simple solutions are not enough.

Virtually all the previous works (see Section 2) investigated the diacritics restoration problem in isolation, i.e., restoring diacritics in otherwise correct texts. This is, however, not realistic: if not enough care and attention are given in typing a text to use proper diacritics, so is, typically, to use the correct spelling. A carefully typed text without diacritics might be more common in the past when Unicode was not widely supported for technical reasons, but this is no longer the case. And crucially, it is neither easy to correct typos before restoring diacritics, as those are not proper texts, nor after, as diacritics would not be restored on mistyped words. If in addition to the missing diacritics other typographical errors are introduced (as is common with fast sloppy typing), specialized diacritics restoration algorithms break down.

Considering these limitations and trends in the current state-of-the-art, we take an approach with these main contributions:

- In contrast to the current state-of-the-art, we use the latest universal sequence-to-
sequence byte-level transformer model ByT5 [8] that has no task- or language-specific structure, vocabulary, or character set.

- We experimentally investigate the effectiveness of this universal method in restoring diacritics on a standard set of 12+1 languages, comparing it to the state-of-the-art.
- We experimentally investigate the effectiveness of this universal method in correcting typos simultaneously with restoring diacritics on the same set of 12+1 languages.

The rest of this paper is organized as follows. We provide a review of related work in the literature on diacritics restoration and typo correction in Sections 2 and 3 respectively. In Section 4, we give a detailed background on our chosen approach and related transformer models in general. In Section 5, we describe the datasets used. In Section 6 we outline the experimental setting and in Section 7, we present the results. Finally, we discuss the findings of this work in Section 8 and summarize them in Section 9.

2 Related Work on Diacritics Restoration

Restoring diacritics is important, as most of the world's languages use and often lose them in the digital age, as discussed above. Thus there are many automatic solutions investigated in scientific literature.

2.1 Classical Approaches

The first approaches were based on rules and simple text statistics.

2.1.1 Rule-Based Approaches

The oldest practicable solutions achieving acceptable accuracy for the diacritics' restoration problem are based on a set of rules. The creation of rules typically requires human intervention and linguistic skills. They also often employ external language resources, i.e., morphological analyzers, syntactic analyzers, and/or morpho-phonological processing tools [9, 10]. The authors in [11] use the lemmatization technique to restore diacritics for the Czech language. Their method contains the set of if-then rules that consider prefixes, suffixes.

As presented in [12], different language resources (i.e., word-based language tri-gram model with the back-off strategy, augmented with the letter-based language model and extremely simple morphological model) can be integrated into a cascade of finite-state transducers to restore diacritics on the Arabic texts. Changing diacritics changes not only the syntax but the semantics of a target (ambiguous) word.

The authors in [13] use a rule-based algorithm to determine the implication relationships between undiacritized and diacritized words by computing distances and conflicts between them based on a distance-map tuned over a long domain experience. Despite handcrafted rules are less flexible to include all aspects of the language and are harder transferable to new domains, they are still in use today 1) when the solving task and domain are very specific; 2) if there is no possibility to get the training corpus of sufficient size and diversity; 3) as the baseline approach or for comparison purposes.

2.1.2 Statistics-Based Approaches

In addition to rule-based approaches, another group that effectively solves the diacritics restoration problems is based on corpus statistics. These methods, in turn, can be further divided into character-level and word-level. The word-level approaches are considered to be a more accurate solution, but they typically rely on expensive resources (i.e., monolingual texts to train language models, dictionaries, etc.) that do not cover the non-standard language forms. All of it makes them more language-dependent and, at the same time, less suitable
for less-resourced languages. On the other hand, character-level approaches more effectively cope with out-of-vocabulary words and therefore can be used to diacritize non-normative language texts (posts on social networks, forums, internet comments, etc.) in which the omitted diacritics problem is especially apparent.

The majority of word-level statistical approaches are based on pre-trained probabilistic \( n \)-gram language models [14]. The \( n \)-gram language models are trained on large monolingual corpora and give a probability of encountering a particular sequence of \( n \) words in a text. The robustness of \( n \)-gram models directly depends on the size and variety of training data. The higher the order \( n \) of the \( n \)-gram model is, the lower perplexity it has and does a better language modeling. Yet high orders \( n \) require a vast amount of data for training and, as a side effect, inflicts sparseness which leads to zero conditional probabilities. The models are usually based on the closed-world assumption, that words not found in the language model do not exist. Therefore smoothing mechanisms become especially important in coping with unseen words (typically assigning non-zero probabilities). Larger \( n \)s are more cumbersome to store and compute, and are typically less beneficial for languages with free word order in a sentence: rare combinations make language models very sparse, less robust, and therefore require pruning.

Since longer sequences are less probable, word-level diacritization approaches often allow back-off or interpolation procedures. The authors [15] successfully apply their language modeling method to the lowercased Slovak texts. The method compares the surrounding context of the target (undiacritized) word with related \( n \)-grams (with \( n = 4 \)). In this way, the method considers 3 preceding and 3 following words around the target one. If the 4-gram is not found, the process continues by backing off to trigrams, bigrams, and, if necessary, to unigrams. The whole diacritization process is iterative and sequential: after the diacritized equivalent for some targeting word is determined, the new target is set.

A similar method is applied to the Igbo language [16]. The authors tested bigram and trigram language models with the back-off strategy and various smoothing techniques experimentally proving the trigram language model with the Add-1 smoothing to be the most accurate for their diacritization problems.

However, the back-off strategy does not always appear to be the best. Experimentally investigated token bigram language model achieved the highest accuracy on the Spanish texts [17]. It outperformed not only the unigram model but a bigram language model with the back-off strategy.

The diacritics restoration problem for Spanish is also tackled in [18] and three different methods are investigated. Their first method relies on the Bayesian framework: the idea behind it is that words closer to the target would give more clues about its correct disambiguation and diacritization. The basis of the second method is the Hidden Markov Model (HMM) method able to solve ambiguity problems by indicating different parts of speech. The third method which is the hybrid of both was able to overcome the limitations of Bayesian (performing poorly on rare words) and HMM (relying on the imperfect morphological analysis) and demonstrate the best performance.

The decision-list approach combines word-form frequencies, morphological information, and collocational information to restore omitted diacritics for Spanish and French languages [19]. First of all, it identifies ambiguity with the help of lexical resources (dictionaries), then collects the context of \( \pm k \) words around the target one. Afterward, it measures collocational distributions (containing the target word) to select the most useful representatives. When log-likelihood values of these collocations are calculated, the algorithm sorts them into decision lists, performs pruning and interpolation. The prepared decision lists are later used to restore diacritics.

The diacritics restoration system for the Croatian language presented in [20] successfully combines the statistical bigram language model with the dictionary (of 750K entries) look-up method. The diacritization process contains three stages. During the first stage substitution schemes applied on the raw text result in generating diacritized candidates; then the validity of each candidate is determined via comparison with dictionary forms; finally, correct forms
are selected with the language model. The authors demonstrate the effectiveness of their method not only on the artificial data (newspaper articles that were undiacritized namely for experiments) but also on the real (forum posts).

The statistical language model can be created not only on the word level but on the character level as in [21]. During the first stage, for recognized words it uses a statistical $n$-gram language model with $n = [1, 4]$ that works on the word level; during the second stage, it processes out-of-vocabulary words with the statistical $n$-gram character-based model that works on the character level. The authors prove that their offered approach leads to the better diacritization accuracy of the Arabic dialectal texts.

2.1.3 Translation-Based Approaches

Sometimes diacritization problem is formulated as the machine translation problem, but instead of translating from the source language to the target, the undiacritized text is translated into the diacritized. However, such a translation problem is less complex due to simpler (one-to-one) alignment and decoding.

The phrase-based Statistical Machine Translation (SMT) system is successfully applied to restore diacritics in the Algiers dialectal texts of the Arabic language [22]. This system uses the Moses (Open Source Toolkit for SMT) engine with the default settings: the bidirectional phrase and lexical translation probabilities; the distortion model with seven features; a word and phrase penalty; and a language model.

The SMT-based method was also applied to Hungarian texts [23]. Similar to [22], Moses was used with the default configuration settings (except for the translation model that contained only unigrams, and the language model with $n$ up to 5), monotone decoding, and without the alignment step. However, SMT alone is not enough for their solving task: agglutinative morphology of the Hungarian language results in plenty of word forms unseen for the system with the restricted vocabulary. To handle it, a morphological analyzer was incorporated into the system. It generates candidates for unseen words that are later fed into the Moses decoder. The probability of each candidate was estimated from the corpus with the linear regression model considering its lemma frequency, the number of productively applied compounding, the number of productively applied derivational affixes, and the frequency of the inflectional suffix sequence returned by the analysis.

Despite the solving problem in [24] being formulated as a word-to-word translation problem, it is not the typical case of SMT. The authors investigated two approaches that only require monolingual corpora. Their lexicon-based approach (applying the most frequent translation observed from the training data) was outperformed by the corpus-based approach (combining information about the probability of translation and the probability of observing a translation in the given context via a simple log-linear model). The research is interesting for several reasons. First of all, the effectiveness of the method is proven in several languages, i.e., Croatian, Serbian, Slovenian. Besides, the diacritics are restored on both standard and non-standard (Web data) texts. Moreover, the authors also performed cross-lingual experiments by training their model on one language and testing on another. The cross-lingual experiments reveal that Croatian and Serbian languages can benefit from each other (training/testing in both directions), whereas the model trained on Slovenian is not effective neither for Croatian nor for Serbian.

2.1.4 Character-Level Approaches

Another important direction in diacritics restoration is character-level approaches. They solve problems that are typically defined as sequence labeling. The iterative process slides through an undiacritized sequence of characters by assigning their diacritized equivalents (labels). Each character is a separate classification instance with the surrounding content as other classification features. Such approaches typically require no additional language tools except for the raw text, which makes them suitable for less-resourced languages. Besides,
Character-level methods are robust when dealing with unknown words. Depending on the chosen classifier, this classification process can be viewed as the independent instance-based classification (assuming that each instance is independent) or sequence classification (considering conditional dependencies between predictions) problem.

The seminal research work in [25] describes the instance-based classification technique applied to the Czech, Hungarian, Polish, and Romanian languages. Authors tested different window sizes (of 1, 3, 5, 7, and 9 lower-cased characters to both sides) with two classifiers: the memory-based approach and the decision tree (C4.5). Their offered method achieved accuracy which is competitive to word-level approaches.

Another research, presented in [26], describes sequence classification tackled with the MaxEnt classifier. This approach is applied to the Arabic language, but instead of pure character features, it employs character- (character n-grams), segment- (words decomposed into prefixes, suffixes, stems, etc.), and part-of-speech tag-based feature types. The successful combination of these diverse sources results in high diacritization accuracy.

Similarly to [25], three instance-based classifiers (decision tree, logistic regression, and Support Vector Machine – SVM) with character n-grams (from a sliding window) as features were investigated for the Hungarian language [27]. The decision tree, which is also good at identifying important features and keeping decisions easy to interpret, was determined to be the most accurate. This research is important for several reasons: it claims the effectiveness of the offered approach on the non-normative language (web data, Facebook posts) and superiority over lexicon lookup (retrieving the most common diacritized forms) and hybrid (lexicon plus character bi-grams) approaches in the comparative experiments. However, comparative experiments are not always in favor of character-level approaches.

In [28] character-level and word-level approaches are compared for the Lithuanian language. The authors used conditional random fields (CRF) as the sequence classifier by applying them to the character-level features. Despite different window sizes (up to 6), the character-based approach was not able to outperform the trigram language model with the back-off strategy. The character-based approach was also not the best choice applied to the Spanish texts [29]: it was outperformed by the decision list (that combines simple word-form frequency, morphological regularity, and collocational information) and part-of-speech tagging (trained on the tagged corpus with information about diacritic placement) approaches.

Two approaches: sequence labeling (i.e., sequence classification) and SMT were compared in [30] for the Tunisian language. The sequence classification approach uses CRF as the classifier applied on the different character (windows up to 6-grams) and word-level (part-of-speech tags of two neighboring words) features. The SMT approach uses Moses with a 5-gram language model and other parameters set to their default values. The comparative experiments demonstrated the superiority of the sequence labeling approach compared to SMT.

Even more comprehensive comparative experiments are performed in [31]: they cover 100 languages and several approaches: lexicon lookup; lexicon lookup with the bigram language model; several character-level methods with various window sizes; the hybrid of the lexicon lookup with the bigram language model (for words in the lexicon) and the character-level approach (for words not in the lexicon). With some exceptions, the hybrid approach performs the best for the majority of languages.

A similar hybrid approach is also successfully applied for the Romanian language [32]. Candidates for each recognized undiacritized target word are generated based on mappings in the dictionary and appropriate ones are selected with the Hidden Markov Model (HMM)-based language model. Diacritics for unknown words are restored with the character-level approach (described in [25]) using windows up to eight characters.

Another hybrid approach that is used for completely different purposes (to clarify/claim the output of the character-based method), is presented in [33] for the Turkish language. During the first stage, it performs sequence classification with the CRF method, but next to current/neighbor ing characters it also uses current/neighbor ing tokens as features: i.e., five character-level and two word-level features. The output of the first stage is fed into the mor-
The authors compared their hybrid approach with several others (rule-based; rule-based with the unigram language model; character-based, but without language validator stage) and proved it is the best.

In contrast to the previously described approaches, the sequence labeling problem can be solved not on the character, but the syllable level as in [34]. The authors solve the instance-based classification problem (by treating each syllable as a separate independent classification instance) and applying the SVM classifier on top. They use different types of features: n-grams of syllables (surrounding the target one with window sizes of 2 and 3); syllable types (uppercased, lowercased, number, other) characterizing surrounding syllables, and dictionary-based features (dictionary words that contain the target syllable). The method achieves high accuracy on Vietnamese texts.

2.2 Deep-Learning-Based Approaches

With the era of Deep Neural Networks (DNNs), the diacritics restoration problem is being solved with these innovative techniques. Some of them rely on word embeddings, i.e., learned word representations capable to capture the context.

Word2vec embeddings were integrated into a 3-stage diacritics restoration system for Turkish in [7]. During the first stage, candidates are generated for the target word. During the second, the morphological analyzer checks if candidates are legitimate words. During the last stage, the word2vec-based tool evaluates the semantic relation of each candidate to its neighboring words with the similarity method and chooses the most suitable one. The authors tested two types of word embedding models (i.e., continuous bag-of-words – CBOW, predicting target word based on its context, and skip-gram, predicting surrounding words based on the input word) and several similarity measures (Cosine, Euclidean, Manhattan, Minkowski, Chebyshev). Their experimental investigation revealed that the skip-gram and cosine similarity approach is most accurate on the Twitter data.

The omitted diacritics problem can also be tackled at the character level and solved as the character classification problem. An example of such a system is for the Arabic language and the core of it is the Bidirectional Recurrent Neural Network (BiRNN) [35]. The BiLSTM takes the undiacritized character (as an input) and outputs its diacritized equivalent (as a label). Input characters are represented as real-number vectors randomly initiated at the beginning and updated during training. The output is the n-dimensional vector with the size of n equal to the size of the output alphabet. The offered approach outperformed the other methods in the comparative experiments. A similar approach is offered for the Hebrew and the base of it is the two-layer LSTM [36].

The Deep Belief Network (DBN) (as a stack of multiple restricted Boltzmann machines in which each layer communicates with both the previous and subsequent layers; but the nodes in each layer don't communicate with each other literally) on the character level is applied to the Arabic [37]. The advantage of DBN compared to the RNN-based approaches is that it overcomes the limitations of backpropagation. The authors tested their approach on several benchmark datasets and compared it to other competing systems claiming their approach to be the best for the diacritization problem.

The robustness of sequence classification is also tested for Croatian, Serbian, Slovenian, and Czech [38]. However, this language-independent part has the additional integration of the 2, 3, 4, 5-gram language model. This language model-based version for the inference uses the left-to-right beam search decoder that combines the neural network and language model likelihoods. Authors compared their method with other approaches (lexicon-based, corpus-based) and systems demonstrating the superiority over the others.

The authors in [39] also assume that the pure character information is not enough to achieve high accuracy for Arabic because lexical, syntactic information is closely interrelated. Due to this reason, they offer the multi-task approach which jointly learns several NLP models namely for segmentation (operating at the character level), part-of-speech tagging, and syntactic diacritics restoration (operating at the word level). All these aggregated models
later are used for diacritics restoration. Segmentation, part-of-speech tagging, and syntactic diacritization models use separate BiLSTM models with the softmax on top of each. Their outputs are aggregated and become the input for the diacritization model which again is one more BiLSTM. The authors compare their model with the other popular approaches and claim its statistically significant improvement.

A similar character classification problem is solved in [40] for the Romanian language. The architecture of their offered system has three different input paths: for characters (to represent the window of characters around the target one), words, and sentences (in which the target character appears). The character input path is represented with BiLSTM encoder for character embeddings; word input – with the FastText word embeddings; sentence input – with the BiLSTM encoder applied on concatenated FastText word embeddings. The authors test their approach with different combinations of input paths (only character input, character input with the word input, etc.) proving that the best accuracy can be reached only with all three input paths.

The sequence classification tasks are also solved for Arabic, Vietnamese and Yoruba languages [41]. The authors tested Temporal Convolutional Network (TCN) (in which information flows from the past-to-the future as in LSTM) and Acousal TCN (A-TCN) (information flows in both directions as in BiLSTM) approaches and compared them to the recurrent sequential models, i.e., LSTM and BiLSTM. The A-TCN approach yielded significant improvement over the TCM and competitive performance to BiLSTM. The hybrid approach (as the 3-stage stacked pipeline) for the Arabic language [42] integrates a character classifier as the first language-independent component. The other two components: character-level deterministic rule-based corrector and word-level statistical corrector are already language-dependent, but help to increase the accuracy even further.

Another research direction to the diacritics restoration problem is sequence-to-sequence (seq2seq) methods. The seq2seq architecture consists of an encoder (converting an input sequence into a context vector) and decoder (reading the context vector to produce an output sequence) blocks as separate DNNs.

Such a seq2seq approach with the RNN-based core was successfully applied to the Turkish [43] and with the LSTM-based core to Vietnamese texts [5, 44]. In [45] for Romanian authors investigated four different encoder-decoder architectures operating on the character level: one-layer LSTMs, two types of stacked LSTMs, CNN-based (3-layer CNN with the concatenated output of encoder and decoder processed with another 2-layer CNN), and determined the CNN-based approach to be the most accurate. Besides, they compared their seq2seq approaches with the classification-based. The first one is a hybrid of BiLSTM (operating on the word level) and CNN (operating on the character level); the second is described in [38] and requires additional language resources (language model). The comparative experiments revealed the superiority of seq2seq methods.

2.2.1 Transformer-Based Approaches

The state-of-the-art techniques in the diacritics restoration, as in all NLP fields, employ transformer-based models.

The multilingual BERT was successfully applied to 12 languages (Vietnamese, Romanian, Latvian, Czech, Polish, Slovak, Irish, Hungarian, French, Turkish, Spanish, and Croatian) [46]. BERT embeddings created on the undiacritized text are fed into a fully connected Feed-Forward Neural Network (FFNN). The output of such a network is a set of instructions (as labels) that define the diacritization operation necessary for each character of the input token. The authors claim their BERT-based approach outperforms all previous state-of-the-art models.

The authors in [47] solve the character classification problem for Vietnamese by offering a novel Transformer Decoder method with the Penalty layer (TDP). The model is a stack of six decoder blocks. The encoder part is redundant since each input character corresponds to only one output character. The penalty layer restricts the output with only possible characters for
the input one. The authors also performed comparative experiments proving their approach is superior to the one offered in [38].

Another transformer-based technique was applied to 14 languages (Bosnian, Czech, Estonian, Croatian, Hungarian, Lithuanian, Latvian, Polish, Romanian, Slovak, Slovenian, Albanian, Serbian, and Montenegro) [48]. The core of the diacritization approach is the Marian Neural Machine Translation (NMT) system with six encoder-decoder layers, which is applied to the frequently occurring character sequences. The research is especially interesting because it is performed in monolingual (training and testing on the same language) and multi-lingual (by either mixing data of all languages or mixing the data of all languages but inserting language codes as the first token of each segment) settings. The authors experimentally determined that monolingual experiments gave almost the same accuracy as multilingual with the language codes.

3 Related Work on Correcting Typographical Errors

A typographical mistake is an error that occurred while printing the material. Historically, this was due to errors in the setup of manual type-setting. The term includes errors caused by mechanical failure or slipping of the arm (or finger) but does not include errors caused due to ignorance, such as spelling errors. However, typos are the subset of a bigger category of misspelling errors. These are of the same importance and are solved with the same methods. The only difference is that typographical errors are easier to model as they depend only on the keyboard (we discuss it more in subsection 5.1) and not the language.

Most classical spelling error correction systems follow these steps:

1. Error detection,
2. Candidate generation,
3. Error correction.

We will cover separate methods constituting this pipeline below.

3.1 Non-Word Detection

Dictionary is the most popular error detection method, sometimes called a lexicon or unigram language model. Dictionary detects non-words: the ones, that cannot be found in it. The first system by date [49] used exactly this method with some additional heuristics. Modern spell checkers, such as GNU Aspell [50] and Hunspell [51] also compare each word of a text to their large lists of words. In Hunspell’s case, the dictionary is more compact by keeping only main word forms with transformation rules, prefixes and suffixes, thus supporting many languages with rich morphology.

There are some downsides to the dictionary method. As noted in [52], about 40% of spelling errors are real-word errors (i.e., “from” → “form”) and can not be detected by the dictionary. [53] showed that GNU Aspell corrects only 51% of errors and is performing best on non-word ones. Secondly, the dictionary cannot cover rare words, such as proper names, country and region names, technical terms, and acronyms. This issue could be dealt with by enlarging the dictionary. However, [52] argues that eventually most of the misspellings would match rare words and therefore fail to be spotted.

3.2 Candidate Generation

This is the task of finding the confusion set of real words for a misspelled one. One can manually craft a confusion set or look for a publicly available one, such as [54] for Chinese. Yet usually sets are generated on the fly. Similarity measure of words is obtained by phonetic or the Minimum Edit Distance algorithms.
The most known phonetic algorithm is Soundex [55, 56]. The cornerstone idea of the Soundex approach is that homophones (the same sounding words) are encoded similarly so that they can be matched regardless of subtle differences in their spelling. A Soundex code is computed from the misspelling, and words that have the same code are retrieved from the dictionary as correction candidates. A similar principle of misspelling encoding was used in the first system bydate [49]. Nowadays Metaphone representations of words (an improvement over Soundex) [57] are used in Aspell [50].

The Minimum Edit Distance [58] measure is defined by a minimum number of edit operations needed to transform one string to another. As reported in [59], more than 80% of errors differ from the correct word by only a single letter, thus this distance between them is low. There are several different edit distance algorithms: Levenshtein [60] (number of insertions, deletions, and substitutions), Damerau-Levenshtein [59] (treating transposition as a single edit), Hamming [61] (number of characters that differ between two equal-length strings), and Longest Common Subsequence [62]. As an example, widely used Aspell uses Damerau-Levenshtein distance between Metaphone representations of words.

### 3.3 Using Context and External Datasets

Given candidates can be simply ranked by their pre-computed distances. On the other hand, some additional information, whether from nearby words or from additional corpus, can aid during target word selection.

The approach in [63] uses a Bayesian combination rule to rank given candidates. First, probabilities for substitution, insertion, and other errors are collected from a corpus of millions of words of typewritten text. Then, given a misspelled word, each inflection and resulting word probabilities are combined to produce a probability estimate for each candidate.

n-gram language models [14] trained on large external corpus can give a conditional probability of how likely a sequence of words is followed by a certain word. n-gram model ranking for confusion sets is used in multiple works for spelling correction [64, 65, 66, 67, 68, 53]. Character-level n-gram also allows calculation of distance measure (such as Hamming in [69]) by comparing character n-grams between two strings [70]. Spelling correction systems using n-gram usually employs Back-off [64, 65, 67] or other [71, 72] smoothing techniques, and sometimes due to its size even require a complex distributed setting [67, 73]. Extensions and problems with n-gram models we already discussed in more detail in Section 2.1.2.

External datasets are especially well exploited by neural network approaches. Authors of [74, 75] use a FastText [76] shallow neural model to learn both known and unknown word vectors as a sum of character n-gram embeddings. Candidate words can then be scored with cosine similarity to the context word vectors. The differences between these two works are text domains: [74] was trained in Bangla language, while [75] on English and Dutch clinical texts.

The ability to learn from vast text resources eventually culminated in the state-of-the-art transformer models, discussed in Sections 3.5 and 4.2.

### 3.4 Real-Word Errors

We already reviewed techniques for detecting and correcting non-word typos. The other, and the far more difficult, group is the real-word errors. These are misspellings that result in other real words. Ironically, these errors are also caused by automatic spelling correction systems [77]. As it is harder to apply unsupervised methods such as the dictionary, there is also a challenge to build tools for different languages with different alphabets and rules [78].

Detection of real-word errors can be done by searching every word in a confusion set and checking for a better alternative [79, 65, 71, 80]. Candidate population is usually done by the n-gram method and others, already discussed in Section 3.2. Some works employ natural language parsers that check grammar [81, 82] or look for semantically unrelated words to their context that have semantically related spelling alternatives [83]. Since detection is similar to
the selection of candidates here, real-word error correction systems often do detection and correction at once.

3.5 Transformer Models for Spelling Error Correction

Recent advances in natural language processing, particularly the transformer architecture [84], solve many problems encountered in traditional approaches. Firstly, the traditional detect-suggest-select pipeline is discarded. Whether it is seq2seq translation or encoder-type each token classification, target words are generated immediately. Secondly, the segregation of non-word and real-word methods is gone here. And finally, the use of context from the whole input sequence and knowledge from additional datasets are now employed. Despite the advantages, some open issues are still being solved.

An important problem for seq2seq models is over-correction: the attempts of a model to correct the sentence even if it is not confident. The authors of [85] addressed this problem for their Korean spelling error correction system by using a dedicated Copy Mechanism. Correction is attempted only if it detects that the input is incorrect, otherwise, the input sequence is copied. The results showed that such a mechanism resulted in better overall performance. Authors of [86] found that over-correction can be mitigated by allowing the transformer to be trained with unfiltered (containing gibberish samples) inputs. This way the model is forced to stick to the initial input unless there is a high certainty of a typo. There is also an attempt to use an additional error detection classification head in the encoder-type transformer model [87].

Usually, small available datasets are not enough to train transformer models. As a result, most works resort to artificial spelling generation. Authors of [86] use statistics of their private 195k sample dataset to generate one of 94M examples. [85] use Grapheme-to-Phoneme and Alphabetical (insertions, deletions, substitutions) generators together with 45 711 private samples. [87] construct random rule-based generator covering the most often error categories of the Vietnamese language. Works utilizing BERT [88] encoder can utilize or supplement the default masking [MASK] token: [89] also use related words from confusion sets while [90] replace with phonologically and visually similar ones.

Original BERT [88] transformer model uses subword tokenization. As misspellings happen at a character level, it is wise to also incorporate characters or other phonetic features. [87] use additional character-level encoder to output character-level vectors. These are concatenated with word embeddings and used in the final word encoder. For the Chinese language, [90] additionally add phonetic and shape embeddings, acquired from separately-trained single-layer GRU [91] networks. Parallel to character classification, authors also perform pronunciation prediction. Similarly, other works for Chinese find it useful to predict not only character but also pinyin and radical – a total of three classification heads. In contrast to these approaches, we use the fine-grained model in the first place and therefore can avoid additional incorporation of character information.

4 Our Methodology

The related work analysis revealed research performed under very different experimental conditions, which makes the results hardly comparable. Different languages have a different level of complexity and ambiguity, and omitting the diacritics or introducing typos exacerbates this problem even more. The training-testing texts cover normative (fiction, periodical, Bible texts) and non-normative (tweets, comments) language types. Investigated approaches are affected by the availability of language resources and the emergence of new methods, and vary from rule-based, traditional machine learning to the most innovative deep learning solutions. There are different evaluation types: extrinsic – evaluating on the downstream tasks vs. intrinsic – calculating the percentage of correctly restored words or characters); different evaluation metrics cover word-level and character-level (including all characters or only with diacritics) techniques. Hence, there being no consensus about which approach is the best for
the diacritics restoration and typographical error correction problems, recent trends suggest that innovative approaches, such as transformer models, are still needed and should be the most promising.

4.1 Formal Definition of the Solving Task

Let \( X = \{x_1, x_2, \ldots, x_N\} \) be a sequence of tokens, constituting our text without diacritics and/or with typos. Let \( Y = \{y_1, y_2, \ldots, y_M\} \) be a sequence of equivalents with their diacritics and/or typos corrected. Depending on the chosen tokenization form, a token can represent a word, subword, character, or byte value.

The function \( \eta \) correctly maps \( X \rightarrow Y \). Our task is to find the method \( \Gamma \) which is as close to an approximation of \( \eta \) as possible.

In this work, we use a transformer model as a method \( \Gamma \). Below we further explain what is behind tokens in our case and how the sequence mapping is performed.

4.1.1 Tokens

Generally, the text is represented as a Unicode string. It is a sequence of code points, which are numbers from 0 through 1114111. For example, the letter “ś” has a code point 115 while the same letter with the additional caron “š” is at 353. Unicode describes a huge amount of various symbols but is very wasteful of memory space. The most popular symbols are at the beginning of this list but still, would have to be represented as 32-bit integers. Instead, UTF-8 encoding is employed to translate Unicode sequence into 8-bit bytes. If the code point is larger than 127, it’s turned into multiple bytes with values between 128 and 255. So the code point 353 of the letter “š” is translated into two bytes 197 and 161 while the letter “ś” retains byte 115. [8] showed better results using a transformer model ByT5 at this byte-level tokenization rather than on characters. Inspired by their success on transliteration and noisy text tasks, we also use the same byte-level tokenization.

4.1.2 Mapping X to Y

One should note that the transformer model does not map the whole target sequence instantly. Starting with the first artificial start token \( y_0 \), it estimates the probability for each next token by taking into account the whole input sequence and previously generated tokens (the context). The probability that the next token is \( y_i \) can be written as

\[
P(y_i | \{x_1, x_2, \ldots, x_N\}, \{y_0, y_1, \ldots, y_{i-1}\}).
\] (1)

Thus the output from a transformer model is a list of probabilities for each token in a vocabulary to be the next token \( y_i \).

The choice of the next token, given the probabilities of all candidates, depends on the decoding algorithm. There are two groups of them: maximization-based (greedy and beam search) and sampling. The most obvious greedy approach is to select a token with the highest probability. During beam search, a defined number (so-called beam size) of word sequences with the highest overall probabilities are kept. This way, a single low-probability word would not shadow a high-overall-probability sequence. Stochastic approaches are inappropriate for our task as there is only one right way to restore diacritics or correct typos.

4.2 Transformer Models

There are several key reasons why transformer [84] architecture became the top-performing in multiple natural language processing leaderboards such as SuperGLUE [92]. The first is that, compared to previous recurrent ones, it is highly parallelizable. It does not need to wait to finish calculations for the previous word. Instead, calculations for all words are done at once. Models can be elementary trained on multiple dedicated machines (such as GPUs) thus
quickly digesting vast amounts of data. Secondly, only after a single block (usually called a layer), the information between all tokens is already exchanged. This is accomplished by a self-attention layer inside the block, which processes a sequence by replacing each element with a weighted average of the rest of the sequence. As there are usually more than five blocks, it allows quick learning of long-range dependencies. Finally, it is less computational power demanding for shorter sequences, which is the case for most of the language tasks. These reasons allowed transformer architecture to flourish.

The capabilities of these models come with a price. Training them from scratch requires dedicated hardware (i.e., a GPU with a large enough memory), takes a long time, and consumes a lot of electricity. Solutions to alleviate this burden started with the introduction of the BERT [88] transformer. This model is pre-trained with a general word masking task to be later fine-tuned for any desired task (the process called transfer learning). It is estimated that pre-training BERT caused more than 300 kg of CO₂ emissions [93], but it can be easily fine-tuned for a custom purpose at a small fraction of that cost. Three years later, now there are plenty of similarly pre-trained publicly available models (e.g., at HuggingFace transformers library [94]). We also build our work on top of one such pre-trained ByT5 [8] model.

In general, transformer models can be grouped into three categories: auto-encoding, auto-regressive, and sequence-to-sequence. We will cover them in more detail below.

4.2.1 Auto-Encoding Transformer Models

This version of the transformer model possesses only an encoder part. It encodes the input text into distinct output vectors for each given token. Attention layers can access all the words in the initial sentence to get the most representative information of the whole sequence. Additional “heads” can be placed on top to further process this representation for a sentence or word classification, extractive question answering, regression, or other tasks. The most popular model of this category is BERT [88].

Several diacritics restoration works use transformer encoders. [46] performs classification on which transformation, described by a diacritic sign and the position in a word, to apply. While [47], although named decoder, has attention masking removed and classifies output diacritic mark categories for each input character.

4.2.2 Auto-Regressive Transformer Models

These models possess only a decoder side of the original architecture and its tokens can only attend to the previous ones. Probably the most known example is one of the latest gigantic (175 billion parameters) transformer models: GPT-3 [95]. It is used in practice by finishing sentence beginnings, so-called zero-shot task solving. In this setting, the human must manage to convey all the necessary information for solving the task in the mentioned beginning, such as providing examples of task solutions. Currently, we do not possess access to the latest GPT-3 model, nor believe it can adequately cover languages we use in this work. Although it would be interesting to test its capabilities in unsupervised zero-shot multilingual diacritics and typos correction.

4.2.3 Sequence-to-Sequence Transformer Models

These are encoder-decoder models. In the encoder part, each token can attend to every other. On the decoder side, there are two types of attention happening. The first one is attention to the decoder past inputs, same as in auto-regressive transformer models. The second one is the full attention to the tokens of the encoder. The most straightforward application of this network is translation. Encoder gets only input language tokens while the decoder is fed target language ones and predicts them one at a time. As the diacritics restoration task can be viewed as a translation task, this transformer type is found in several related works [96, 97, 98].
The most popular model of this category is T5 [99]. Authors framed various tasks, even ones including numbers to text-to-text format. They reported that there is no significant difference if a separate “head” is used or an answer is generated as a simple text. This in turn made the model very simple to use. In this work, we use the follow-up multilingual ByT5 [8] model designed to work with byte-level tokens. We think that the seq2seq approach is the most adequate as it is universal. Additionally, operating on byte-level gives a level of immunity to minor text noise, i.e., against typographical errors, and is more language-universal.

4.2.4 ByT5 Model

ByT5 [8] is a general-purpose pre-trained multilingual text-to-text model, based on earlier predecessor mT5 [100]. It completely dispenses of SentencePiece [101] tokenizer, as it does not need any. The authors concentrated 3/4 of the parameters into the encoder by decoupling the depth of the encoder and decoder. A small version of ByT5 now has 12 encoder and 4 decoder layers.

In the ByT5 model case, the total vocabulary size is 384, consisting of 3 special tokens (<pad> for padding, </s> for end of the sequence, and <unk> for unknown), 256 = 2^8 the main 8-bit bytes, and 125 extra sentinel tokens used only in pre-training task. In the small version, the vocabulary accounts only for 0.3% of total parameters while in a similarly-sized mT5 model vocabulary took 85% of total parameters. As a result, the small ByT5 working with fine-granularity tokens (bytes) outperforms mT5, which worked inefficiently due to large-granularity rarely used vocabulary parts (subwords) taking up much parameter space.

Due to its byte-level nature, ByT5 is slower to compute. More fine-grained tokenization produces more tokens for the same text and requires more time for the model to digest. However, ByT5 authors showed that for short-to-medium length text the time increase is negligible. This is the case for diacritics restoration as the input is composed of a single sentence.

4.3 Training Hyperparameters

Artificial neural networks are trained by updating their weights according to their response to the input. In particular, we focus on mini-batch gradient descent. For every mini-batch of \( n \) training examples (input \( x^i \) and output \( y^i \) pairs) model parameters \( \theta \) are updated using an objective function \( J \):

\[
\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{i+n}; y^{i+n}).
\]  

Adam [102] and AdaFactor [103] extensions of this vanilla gradient descent currently are the most prevalent optimization algorithms for transformer models. The success of training of the models depends a lot on the setting of the hyperparameters in (2) correctly, such as the batch size \( n \), the sequence length within a sample, and the learning rate \( \eta \). We will discuss them in more detail.

4.3.1 Batch Size

It is the number of samples to run through the model before updating weights. The more tokens it has, the less disturbance will an individual sample cause during (a much smoother) weight update. On the other hand, very large batches take much time to compute and have diminishing gains.

The first popular pre-trained transformer BERT [88] for classification used a batch size of 256 sequences. A later model RoBERTa [104] showed that an increase of batch size (up to 8k) and dataset size accordingly improves downstream performance. Yet the same authors had to fine-tune downstream applications using only batches of size up to 48.

The popular seq2seq transformer T5 [99] used batch size 128 for both pre-training and fine-tuning. Follow-up models such as the multilingual version mT5 [100], the grammatical
error correction model gT5 [105], and ByT5 [8] (the one we use in this work) all carried on with the same value for fine-tuning. The same size is also used in works solving the diacritics restoration task [47, 106].

In conclusion, we can use a batch size of 128 or greater. All methods of this family use the same size and, moreover, we are not strictly limited by the dataset size to increase it for better performance.

### 4.3.2 Maximum Sequence Length

When choosing the right batch size one should also account for the maximum number of tokens allowed in a sample. There are two caveats here. First, the time complexity of the transformer model is quadratic on sequence length $n$ (number of tokens) $O(n^2)$ thus shorter sequences are preferred for a faster training time. Secondly, the model we use operates in byte granularity and needs more tokens to express the same text compared to word-level granularity models. Authors of ByT5 [8] report that English language sequences in byte tokens are about five times longer than in subword ones. As a result, the maximum sequence length for ByT5 is set to 1024 tokens. In our case, samples are sentences and, in practice, they all fit into this length.

### 4.3.3 Learning Rate

The last important parameter in (2) is the learning rate $\eta$. It controls how much the model parameters have to be updated. Low values of $\eta$ ensure smooth monotonic but small updates and prolonged convergence. On the other hand, the higher learning rates would enlarge improvements and speed up the training. However, due to the higher “energy” (or “temperature”) in the optimization the high $\eta$ causes “bouncing” of parameter values and prevents settling in the best spot, resulting in the higher final training loss. An optimal learning rate value, as used during fine-tuning of the T5 family models [99, 103, 100, 8] with Adafactor optimizer, is 0.001.

Sometimes better results can be achieved by scheduling learning rate values during the training. There is typically the so-called warm-up period in the beginning to level discrepancies between previous parameters and new domain updates. It contains low or linearly increasing values of the learning rate. Similarly, as the training is to be finished, the “energy” of optimization can be lowered by lowering the learning rate and allowing neural network weights to settle in a more favorable position. As an example, during the original T5 [99] pre-training a constant warm-up and following inverse square root decay with a peak learning rate of 0.01 was used. Yet fine-tuning was performed with a constant value of 0.001. Such learning rate is not dependent on dataset size and enables straightforward comparisons of different setups. Overall, learning rate schedules can improve constant learning rate results but are less flexible to experiment with.

### 4.4 Evaluation

To evaluate diacritics restoration capabilities we use the alpha-word accuracy metric from [38]. Each text sample is segmented into words and for each word we check if it is an alpha-word (alphabetical word):

- All characters in the word are alphabetic, those with general Unicode category property being one of “Lm”, “Lt”, “Lu”, “Ll”, or “Lo”;
- It has at least one letter.

Given the number of gold (correct text) words satisfying this condition $T_g$, and the number of these words being correctly predicted by the system $T_s$, alpha-word accuracy is

$$\text{alpha-word accuracy} = \frac{T_s}{T_g} \cdot 100\%.$$ 

(3)
This metric ensures that our results are not polluted by words that cannot have accents (e.g., numbers). Also, it takes into account both occasions of necessary and unnecessary accent generation. Other metrics as Word Error Rate (WER) or Diacritic Error Rate (DER) restrict themselves to $T_g$ of only diacritized letters in the gold standard text [37].

5 Dataset

The popularity of the internet brought many abundant multilingual text resources. They usually vary from noisy and colossal to small quantity but high quality. Good examples of the first side are the Common Crawl dataset of more than 20TB of data and its version OSCAR [107] filtered by language. Such large datasets are now one of the main building blocks of popular transformer model pre-training but are very costly to work with during fine-tuning scenarios like ours. The other extreme, such as a small high-quality Universal Dependencies [108] dataset is just too small to cover most aspects in each language.

Recent works on diacritics restoration seek a compromise between these two extremes. [48] uses OpenSubtitles dataset which is of satisfactory quality. On the other hand, [46] combine low-quality and high-quality datasets. They train first with the noisy web data and finish with the higher quality Wikipedia. However, training took two weeks for each language to reach state-of-the-art results.

We use the same 12-language (Croatian, Czech, French, Hungarian, Irish, Latvian, Polish, Romanian, Slovak, Spanish, Turkish, Vietnamese) Wikipedia dataset proposed in [38]. Recent state-of-the-art diacritics restoration results were reported [46] for this dataset, so it is straightforward to compare with our methods on this particular task. As our focus is on efficiency, we omitted the large web text part to work only with the better-quality Wikipedia part. We also added Lithuanian language to the list, using the tools publicly provided by the original authors of [38].

The dataset consists of training, development, and testing sets. All three are lowercased, tokenized to words, and split into sentences. The split between sets is performed on Wikipedia article level. We show statistics of the train set in Table 1. Test sets do not differ much except that each language has exactly 30000 sentences allocated and thus has a similar amount of words. Percentages of alpha-words, diacritic words, and diacritic letters do not deviate for the test set by more than 10%.

The dataset is already preprocessed to be used by simpler approaches such as dictionary mapping. ByT5 tokenization does not require that as any text can be encoded in UTF-8 bytes. Thus it can work with any processed or unprocessed text.

5.1 Realistic Model of Typos

We produce our pairs of correct (target) and incorrect (input) text by taking the dataset as the correct (gold) text and generating the corresponding incorrect text automatically.

Diacritic removal is straightforward, simply done by replacing all diacritic letters with the non-diacritic equivalents.

However, for typographical error induction, a dedicated realistic corruption model is required. The approach taken by other works [77, 86] is to infer probabilities for each error group from the available smaller dataset and use them to generate errors on the target one. We took the same approach in this work.

There are four prevailing categories of typographical errors. [59, 109] reported that more than 80% of errors can be attributed to substitution, deletion, insertion, or transposition errors. This division allows us to model each category separately.

The physical keyboard layout plays an important role in influencing typos. A single keypress instruction consists of information of which hand, finger, and key row to select. [52] argues that confusion of these instructions is the main culprit of substitution errors, while mixed instructing timing between two hands (operating on different parts of a keyboard) – of
Table 1: Languages and the training datasets statistics. Diacritic percentages are calculated among alphabetical words or letters. Alphabetical words (alpha-words) range from 72% to 86% of all total words including numbers.

<table>
<thead>
<tr>
<th>Language</th>
<th>Diacritic letters</th>
<th>Keyboard family</th>
<th>Sentences</th>
<th>Alpha-words</th>
<th>Words</th>
<th>Letters</th>
<th>Alpha-diacritic %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Croatian</td>
<td>5</td>
<td>QWERTZ</td>
<td>802 610</td>
<td>12 914 186</td>
<td>14.55</td>
<td>2.78</td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td>19</td>
<td>QWERTY</td>
<td>952 909</td>
<td>14 730 260</td>
<td>48.69</td>
<td>12.90</td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>15</td>
<td>AZERTY</td>
<td>1 818 618</td>
<td>37 612 736</td>
<td>16.49</td>
<td>3.72</td>
<td></td>
</tr>
<tr>
<td>Hungarian</td>
<td>9</td>
<td>QWERTZ</td>
<td>1 294 605</td>
<td>17 587 448</td>
<td>50.05</td>
<td>11.48</td>
<td></td>
</tr>
<tr>
<td>Irish</td>
<td>5</td>
<td>QWERTY</td>
<td>50 825</td>
<td>1 005 620</td>
<td>29.52</td>
<td>7.04</td>
<td></td>
</tr>
<tr>
<td>Latvian</td>
<td>15</td>
<td>QWERTY</td>
<td>315 807</td>
<td>4 244 914</td>
<td>48.57</td>
<td>10.27</td>
<td></td>
</tr>
<tr>
<td>Lithuanian</td>
<td>9</td>
<td>QWERTY</td>
<td>612 724</td>
<td>7 096 677</td>
<td>38.75</td>
<td>7.00</td>
<td></td>
</tr>
<tr>
<td>Polish</td>
<td>9</td>
<td>QWERTY</td>
<td>1 069 841</td>
<td>16 178 130</td>
<td>32.71</td>
<td>6.42</td>
<td></td>
</tr>
<tr>
<td>Romanian</td>
<td>6</td>
<td>QWERTY</td>
<td>837 647</td>
<td>16 050 136</td>
<td>27.04</td>
<td>5.87</td>
<td></td>
</tr>
<tr>
<td>Slovak</td>
<td>25</td>
<td>QWERTZ</td>
<td>613 727</td>
<td>9 180 800</td>
<td>42.38</td>
<td>9.32</td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>7</td>
<td>QWERTY</td>
<td>1 735 516</td>
<td>42 863 263</td>
<td>11.50</td>
<td>2.33</td>
<td></td>
</tr>
<tr>
<td>Turkish</td>
<td>11</td>
<td>QWERTY</td>
<td>875 781</td>
<td>10 785 516</td>
<td>31.35</td>
<td>6.30</td>
<td></td>
</tr>
<tr>
<td>Vietnamese</td>
<td>67</td>
<td>QWERTY</td>
<td>819 918</td>
<td>20 241 117</td>
<td>81.18</td>
<td>25.94</td>
<td></td>
</tr>
</tbody>
</table>

transposition ones. While there may be more causes such as visual and phonological factors [110], we restrict ourselves to the physical keyboard layout influence. This allows us to model typographical errors for all languages given the distribution of keyboard errors for a single one. We also make no distinction between physical and touchscreen keyboards, large or small.

There are only limited misspelling resources for the data-rich English language, as shown in Table 2. The largest one is the GitHub Typo corpus [111]. Although it contains edits for multiple languages, only English is of significant size. There is also a multilingual Wikipedia edit history, which could be prepared similarly to the GitHub dataset. However, it must be filtered [112] to not include non-typographical error-related examples. Incorporating the Twitter Typo corpus [113] may also not be worth the effort, as domains are different, as well as the length of text spans (needed to normalize error frequencies). In the end, we used a single GitHub Typo Corpus to derive probabilities of errors.

Table 2: Related datasets for English misspelling correction

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of edits</th>
<th>Collection method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub Typo Corpus [111]</td>
<td>350 000</td>
<td>keyboard</td>
</tr>
<tr>
<td>Twitter Typo Corpus [113]</td>
<td>39 171</td>
<td>keyboard</td>
</tr>
<tr>
<td>Birkbeck spelling corpus [114]</td>
<td>36 133</td>
<td>handwritten</td>
</tr>
<tr>
<td>Holbrook misspelling corpus [115, 116]</td>
<td>1 791</td>
<td>handwritten</td>
</tr>
</tbody>
</table>

More details on generating the typos we provide in Section 6.2.

6 Experiment Details

Here we provide some more details on our experiments.
6.1 ByT5 Model Fine-Tuning

We chose the batch size of 256 and default ByT5 maximum sequence length of 1024. Such configuration matches the total maximum number of tokens \((256 \times 1024 = 2048 \times 128)\) with the last best system for diacritics restoration [46]. The larger sequence length is essential as our model works on byte-level fine-grained tokens compared to coarser subword level models.

We used GeForce RTX 2080 Ti GPU. Due to modest memory size, we employed the gradient accumulation technique. It accumulates gradients in a continuous rather than in a parallel fashion. In addition, feeding only a single sample at a time allowed us to avoid padding. We trained each model for 2048 steps (each consisting of 256 sentences/samples) and it took up to 10 hours for a single model. E.g., for the Lithuanian language (see Table 1) it uses \(2048 \times 256 = 524288\) out of the total 612724 sentences, which corresponds to 0.86 epochs. In our results, we refer to such basic training as the one trained for \(\times 1\) a number of sentences (\(\#\)samples).

We used Adafactor [103] optimizer with a constant learning rate of 0.001. The same setup was employed by ByT5 [8] authors for fine-tuning experiments. Moreover, the Adafactor optimizer also has very little auxiliary storage compared to the other popular optimizer Adam [102]. More complex learning rate schedules may give a bit better performance, but it would be more difficult to compare our runs, so we stick to the constant learning rate approach.

For the diacritics restoration task with each language, we trained three different models. Each model has different weight initialization and data sampling is performed differently, according to a given random seed. Results are reported as a mean and standard deviation over these three runs. In addition, we trained models for simultaneous diacritics and typographical errors correction for each language.

We also trained several models for a much longer time. First, we continued our basic fine-tuning setup with a batch size of 258 up to 6000 steps (all other basic setups are just up to 2048). At this stage loss becomes a bit noisy (although low), so we increased our batch size to 8192 and continued training further. Due to the change in batch size, we report our model training steps by how much training data, compared to our basic setup, it consumed. Particularly, in our results, we report models trained for \(\times 7\) and \(\times 17\) times the number of samples of the basic setup. As such long training is very time-consuming, we performed only a few of them. We think that it still sufficiently indicates the scaling effects.

For text generation, in all our experiments we used a beam size of 2. Yet later runs revealed that there is hardly a difference in size. As a result, for future work, we recommend sticking to a simpler beam size of 1.

The training script and Pytorch model implementation were used from the Hugging Face library [94]. If not stated otherwise, we used all default parameters as in this library version 4.12.0.

6.2 Generation of Typographical Errors

We took a similar approach for the generation of typographical errors as in [77]. Close to a process of text writing, the program moves through each symbol and induces errors in a stochastic manner by evaluating probabilities of various error types for each character. This includes deletion, insertion, substitution, and transposition operations.

The chance for a letter to participate in a particular error type is determined according to the frequency of errors in the reference dataset. We used the largest known original typo dataset: GitHub Typo Corpus [111]. The dataset was filtered for only English language typos and characters were selected with a count of at least 1000. Given the final character set \(C\), a total number of times \(f(c)\) the character \(c \in C\) or a specific typo pattern appeared in the selected corpus, the following probabilities for each character are considered:

\[
P(\text{deletion} \mid c) = \frac{f(c \rightarrow \emptyset)}{f(c)},
\]
Figure 1: Distribution of generated typographical errors by category (the left vertical axis and stacked bars). Proportions for the English part of the GitHub corpus (used to derive generation probabilities) are also depicted for reference. The total percentage of induced corruptions (the right vertical axis and corresponding blue dots).

\[
P(\text{substitution} \mid c) = \frac{\sum_{c' \in C} f(c \rightarrow c')} {f(c)}, \quad (5)
\]

\[
P(\text{insertion after} \mid c) = \frac{\sum_{c' \in C} f(c \rightarrow cc')} {2f(c)}, \quad P(\text{insertion before} \mid c) = \frac{\sum_{c' \in C} f(c \rightarrow cc')}{2f(c)}, \quad (6)
\]

\[
P(\text{transposition} \mid cc') = \frac{f(cc' \rightarrow c'c)} {f(cc')} . \quad (7)
\]

Note that we divide insertion errors into two distinct categories, whether the character is inserted after the one in question, or before. Both insertion probabilities are collected from the same samples, so we divide them by two. An alternative way would be to collect triplets of characters before, the one in question, and after, but probabilities would then be sparse. Nevertheless, our chosen approach covers fat fingers-like situations.

We ran some typographical error induction experiments on the original GitHub corpus and confirmed that our generation method aligns with the original error type distribution. Initially, only about 1% of characters were corrupted so we scaled our probabilities by a factor of three to be close to a low error rate as defined in [77]. The final error type distribution and percentage of corrupted characters for each language are depicted in Figure 1. The amount of generated errors for each language slightly varies because letter frequencies derived from English differ in other languages.
Insertion and substitution errors can result in many different outcomes. Probabilities for specific letters to emerge, given that this type of error occurs at a specific place, are computed by the following equations:

\[
P(c \rightarrow c' \mid c, \text{substitution}) = \frac{f(c \rightarrow c')}{\sum_{c' \in C} f(c \rightarrow c')}.
\] (8)

\[
P(c \rightarrow cc' \mid c, \text{insertion after}) = \frac{f(c \rightarrow cc')}{\sum_{c' \in C} f(c \rightarrow cc')}.
\] (9)

\[
P(c \rightarrow c'c \mid c, \text{insertion before}) = \frac{f(c \rightarrow c'c)}{\sum_{c' \in C} f(c \rightarrow cc')}
\] (10)

As mentioned, we take the typo statistics from the English dataset and run on the assumption here, that typos are based purely on the layout of the keyboard (proximity of keys, etc.), so the same typo statistics will be in all the other languages using the QWERTY layout. We do not have to deal with the extensions of the character sets and keyboard layouts for different languages, as we only introduce typos to the undiacritized versions of the texts, irrespective of case. We disregard possible other minor variations in the keyboard layouts as insignificant.

For the Croatian, French, Hungarian, and Slovak languages, correspondingly to their different keyboard layout family (see Table 1) we remapped the original English QWERTY dataset before inferring probabilities. E.g., for Croatian, having QWER TZ layout, we had to swap letters “z” and “y” when calculating probabilities. In our initial experiments, we did not observe significant model performance differences between the QWERTY and remapped typo generation versions.

7 Results

We present the results of our different experiments here.

7.1 Diacritics Restoration

Diacritics restoration results are presented in Table 3. Our ByT5 method results lie between dictionary (a simple statistical unigram model) and the state-of-the-art [46]. The highest alpha-word accuracy is for French, Spanish, and Croatian, only 0.34%, 0.29%, and 0.56% behind the state-of-the-art, respectively. These languages have the smallest percentage of diacritic words (see Table 1). The lowest scores are recorded for Vietnamese and Latvian at 94.25% and 96.33%, respectively. We also note that the Irish language with the smallest dataset has the highest standard deviation of 0.32%.

The “Raw” column in Table 3 indicates the alpha-word accuracy of the uncorrected text for comparison. Naturally, the more diacritic-heavy the language is the lower the number.

7.1.1 Dict.+ByT5 Approach

We noticed that the dictionary method outperforms ByT5 for words that have only a single target translation in the dictionary. We grouped words by how many translation targets in the dictionary they have and show the ratio of ByT5 to Dictionary error rates in Table 4. Resulting values higher than 1 indicate Dictionary outperforming ByT5. This is the case for all languages at word group with only a single translation.
Table 3: Alpha-word accuracy results (%) for the diacritics restoration task. We report mean and standard deviation for three separate training runs with different initial model weights and dataset samplings, as well as a single seven times longer training run.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>×1</td>
<td>×7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>×7</td>
<td></td>
</tr>
<tr>
<td>Croatian</td>
<td>85.01</td>
<td>99.11</td>
<td>99.73</td>
<td>99.17±0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.42±0.03</td>
</tr>
<tr>
<td>Czech</td>
<td>49.71</td>
<td>95.67</td>
<td>99.22</td>
<td>98.01±0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98.38±0.04</td>
</tr>
<tr>
<td>French</td>
<td>83.11</td>
<td>97.98</td>
<td>99.71</td>
<td>99.37±0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.49±0.03</td>
</tr>
<tr>
<td>Hungarian</td>
<td>50.34</td>
<td>96.22</td>
<td>99.41</td>
<td>98.42±0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.20</td>
</tr>
<tr>
<td>Irish</td>
<td>69.97</td>
<td>96.65</td>
<td>98.88</td>
<td>98.14±0.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98.40±0.16</td>
</tr>
<tr>
<td>Latvian</td>
<td>50.14</td>
<td>90.59</td>
<td>98.63</td>
<td>96.33±0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97.78</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>60.76</td>
<td>93.83</td>
<td>—</td>
<td>97.94±0.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.07</td>
</tr>
<tr>
<td>Polish</td>
<td>66.73</td>
<td>97.00</td>
<td>99.66</td>
<td>99.00±0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.16±0.02</td>
</tr>
<tr>
<td>Romanian</td>
<td>70.37</td>
<td>96.09</td>
<td>98.64</td>
<td>97.99±0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98.17±0.04</td>
</tr>
<tr>
<td>Slovak</td>
<td>56.34</td>
<td>96.88</td>
<td>99.32</td>
<td>98.43±0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98.77±0.02</td>
</tr>
<tr>
<td>Spanish</td>
<td>87.97</td>
<td>99.11</td>
<td>99.62</td>
<td>99.33±0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.43±0.02</td>
</tr>
<tr>
<td>Turkish</td>
<td>68.39</td>
<td>98.41</td>
<td>98.95</td>
<td>98.86±0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.03±0.02</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>15.88</td>
<td>73.53</td>
<td>98.53</td>
<td>94.25±0.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97.53</td>
</tr>
<tr>
<td>On average</td>
<td>62.67</td>
<td>94.70</td>
<td>99.19</td>
<td>98.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>98.32</td>
</tr>
</tbody>
</table>

Table 4 also portrays how the ratio of ByT5 to Dictionary error rates changes during the half and full training. The trend is obvious – transformer is getting better for all word groups with training. If our training would be much longer, ByT5 may even surpass Dictionary at word group of one translation candidate. This is exactly what happened for Latvian and Lithuanian languages after seven times more training samples.

Note, that at half training the standard deviation of the Turkish ratio is abnormally high. This is due to one of three ByT5 training runs temporarily failing. Yet during further training, the run recovered up to the same accuracy level as the other two. This is a good example of how different training dynamics can be depending on different initial conditions and different data sampling.

We constructed a hybrid approach by letting Dictionary restore words with only a single translation candidate while leaving all the other words for transformer. For our standard training, this improved single ByT5 results by up to 0.37% on average and allowed us to reach state-of-the-art results for the Turkish language. However, we can observe, that with longer training the pure ByT5 model can catch up or even surpass the hybrid approach.

7.2 Simultaneous Diacritics and Typos Correction

Results of simultaneous diacritic and typographic error correction are represented in Table 5. It is clear that alpha-word accuracy results are significantly lower across the board, compared to just restoring diacritics.

The dictionary here was used the same from the previous experiment, i.e., “trained” on the typo-free diacritization-only task, in both the standalone and hybrid approaches.

Reduction for ByT5 on average is by 7.84%, while for hybrid Dict.+ByT5 approach by 3.71%. Smaller reduction for hybrid method suggests that transformers do not cope well with the same words that it successfully dealt with when there were no typos present. A possible reason may be that more learning is required by both tasks, and only up to 10 hours of training might not be enough. Training the Hungarian model up to 17 times longer improves
Table 4: Alpha-word error ratio between ByT5 and Dictionary methods for two word groups and models in different training stages. The values higher than 1 indicate that the Dictionary method restores diacritics better. The first word group corresponds to words with exactly one possible translation target, the second — two. Groups are determined by the training set statistics while results are reported on the testing set.

<table>
<thead>
<tr>
<th>Language</th>
<th>One dictionary candidate</th>
<th>Two dictionary candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#samples: $\times 0.5$</td>
<td>$\times 1$</td>
</tr>
<tr>
<td>Croatian</td>
<td>6.37±0.98</td>
<td>4.98±0.52</td>
</tr>
<tr>
<td>Czech</td>
<td>4.74±0.19</td>
<td>3.53±0.03</td>
</tr>
<tr>
<td>French</td>
<td>5.29±0.17</td>
<td>4.98±0.48</td>
</tr>
<tr>
<td>Hungarian</td>
<td>7.35±0.48</td>
<td>4.37±0.10</td>
</tr>
<tr>
<td>Irish</td>
<td>2.13±0.21</td>
<td>2.27±0.84</td>
</tr>
<tr>
<td>Latvian</td>
<td>2.43±0.17</td>
<td>1.77±0.11</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>2.61±0.18</td>
<td>2.00±0.36</td>
</tr>
<tr>
<td>Polish</td>
<td>4.06±0.24</td>
<td>2.56±0.15</td>
</tr>
<tr>
<td>Romanian</td>
<td>3.66±0.44</td>
<td>2.63±0.12</td>
</tr>
<tr>
<td>Slovak</td>
<td>4.29±0.03</td>
<td>3.00±0.21</td>
</tr>
<tr>
<td>Spanish</td>
<td>5.4±0.54</td>
<td>4.18±0.57</td>
</tr>
<tr>
<td>Turkish</td>
<td>10.5±10.84</td>
<td>2.70±0.24</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>2.6±0.10</td>
<td>2.38±0.20</td>
</tr>
</tbody>
</table>

the performance substantially, but the gap of 2.83% between the ByT5 model and the hybrid remains.
Table 5: Alpha-word accuracies for the simultaneous diacritics and typographic error correction.

<table>
<thead>
<tr>
<th>Language</th>
<th>Raw</th>
<th>Dict.</th>
<th>ByT5 #samples: ×1</th>
<th>Dict.+ByT5 #samples: ×1×17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Croatian</td>
<td>64.05</td>
<td>74.06</td>
<td>90.27</td>
<td>96.71</td>
</tr>
<tr>
<td>Czech</td>
<td>38.68</td>
<td>71.37</td>
<td>89.88</td>
<td>94.52</td>
</tr>
<tr>
<td>French</td>
<td>60.81</td>
<td>70.87</td>
<td>93.45</td>
<td>96.52</td>
</tr>
<tr>
<td>Hungarian</td>
<td>38.16</td>
<td>69.84</td>
<td>88.31 ×93.87</td>
<td>94.31 ×96.85</td>
</tr>
<tr>
<td>Irish</td>
<td>53.49</td>
<td>73.16</td>
<td>89.48</td>
<td>94.49</td>
</tr>
<tr>
<td>Latvian</td>
<td>37.69</td>
<td>66.29</td>
<td>88.88</td>
<td>93.01</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>44.78</td>
<td>68.44</td>
<td>89.68</td>
<td>94.70</td>
</tr>
<tr>
<td>Polish</td>
<td>49.10</td>
<td>70.02</td>
<td>91.38</td>
<td>96.76</td>
</tr>
<tr>
<td>Romanian</td>
<td>51.93</td>
<td>70.29</td>
<td>90.50</td>
<td>94.14</td>
</tr>
<tr>
<td>Slovak</td>
<td>43.92</td>
<td>72.89</td>
<td>91.05</td>
<td>95.56</td>
</tr>
<tr>
<td>Spanish</td>
<td>64.07</td>
<td>71.58</td>
<td>93.12</td>
<td>95.98</td>
</tr>
<tr>
<td>Turkish</td>
<td>51.18</td>
<td>72.58</td>
<td>90.00</td>
<td>95.29</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>11.92</td>
<td>56.19</td>
<td>87.34</td>
<td>87.86</td>
</tr>
<tr>
<td>On average</td>
<td>46.91</td>
<td>71.89</td>
<td>90.26</td>
<td>94.60</td>
</tr>
</tbody>
</table>

7.3 Performance on the Zipf’s Tail

Word frequencies can be modeled reasonably well by a Zipf distribution. It is a very heavy-tailed distribution: there is a vast number of words with low frequencies. The abundance of such words is a challenge for most learning systems as the data for these points is sparse. Our question is, how hard are these words for our trained models?

We grouped words that were in our test set by their frequencies in the train set. The resulting word groups are:

- **Unseen**: present in the test but not in train data;
- **(0, 100]**: words appearing in training set from 1 up to 100 times;
- **(100, 10000]**: words appearing in training set from 101 up to 10000 times.

Alpha-word accuracy results for these groups are shown in Table 6.

A substantial part of errors come from the words unseen during the training. Excluding Vietnamese and Irish, it ranges from 13% (Spanish, French) up to 36% for Slovak. Vietnamese outlier of 1% may be due to linguistic nature while Irish one of 46% is due to the very small dataset. Overall, the smaller the dataset (Table 1), the more unseen or rare words and associated errors we have.

Similar to the dictionary and other classical methods, unseen data is also a significant source of errors for the transformer model. Differently from classical approaches, however, the transformer model is based on neural networks and can generalize to unseen data. To investigate this generalization, we filtered all the words that were in the test set and not in train and calculated percentages as is shown in Table 7. We can see that ByT5 successfully restores more than 76% of unseen words for each language.
Table 6: Distribution of diacritics restoration errors for the different frequencies of the words in the training dataset: the first three columns. The intervals indicate the bounds of how many times the words in this group were encountered in the training dataset. The last two columns indicate the percentages of how much of the training dataset was constituted by these word groups.

<table>
<thead>
<tr>
<th>Language</th>
<th>Percentage of all errors</th>
<th>% of train dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unseen</td>
<td>(0, 100]</td>
</tr>
<tr>
<td>Croatian</td>
<td>31.42±1.65</td>
<td>54.23±1.27</td>
</tr>
<tr>
<td>Czech</td>
<td>20.42±0.24</td>
<td>51.07±1.31</td>
</tr>
<tr>
<td>French</td>
<td>13.55±0.88</td>
<td>42.29±3.26</td>
</tr>
<tr>
<td>Hungarian</td>
<td>26.78±0.33</td>
<td>49.96±0.59</td>
</tr>
<tr>
<td>Irish</td>
<td>46.51±6.04</td>
<td>35.37±0.65</td>
</tr>
<tr>
<td>Latvian</td>
<td>25.94±0.95</td>
<td>49.87±0.16</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>26.13±0.60</td>
<td>47.93±1.03</td>
</tr>
<tr>
<td>Polish</td>
<td>18.31±0.52</td>
<td>48.49±0.89</td>
</tr>
<tr>
<td>Romanian</td>
<td>16.06±0.29</td>
<td>43.46±0.96</td>
</tr>
<tr>
<td>Slovak</td>
<td>36.68±1.00</td>
<td>52.96±0.29</td>
</tr>
<tr>
<td>Spanish</td>
<td>13.22±0.24</td>
<td>39.29±1.58</td>
</tr>
<tr>
<td>Turkish</td>
<td>24.11±0.48</td>
<td>54.03±1.13</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>1.48±0.01</td>
<td>7.85±0.19</td>
</tr>
<tr>
<td></td>
<td>×7 #samples</td>
<td>3.37</td>
</tr>
</tbody>
</table>
Table 7: Confusion matrix of unseen words diacritics restoration performance of the ByT5 model. Unseen words with and without diacritics are presented separately. The last column depicts the total number of unseen words for each language.

<table>
<thead>
<tr>
<th>Language</th>
<th>With diacritics, %</th>
<th>Without diacritics, %</th>
<th>Total unseen</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Failed</td>
<td>Restored</td>
<td>Failed</td>
</tr>
<tr>
<td>Croatian</td>
<td>6.8±0.2</td>
<td>15.9±0.2</td>
<td>4.3±0.4</td>
</tr>
<tr>
<td>Czech</td>
<td>16.4±0.2</td>
<td>37.7±0.2</td>
<td>5.0±0.4</td>
</tr>
<tr>
<td>French</td>
<td>10.1±0.1</td>
<td>9.8±0.1</td>
<td>4.6±0.3</td>
</tr>
<tr>
<td>Hungarian</td>
<td>10.1±0.2</td>
<td>58.2±0.2</td>
<td>2.3±0.1</td>
</tr>
<tr>
<td>×7 #samples</td>
<td>7.1</td>
<td>61.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Irish</td>
<td>12.1±0.5</td>
<td>25.6±0.5</td>
<td>5.3±1.0</td>
</tr>
<tr>
<td>Latvian</td>
<td>16.8±0.7</td>
<td>39.9±0.7</td>
<td>5.5±0.6</td>
</tr>
<tr>
<td>×7 #samples</td>
<td>13.6</td>
<td>43.1</td>
<td>3.9</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>8.1±0.4</td>
<td>29.7±0.4</td>
<td>4.4±1.4</td>
</tr>
<tr>
<td>×7 #samples</td>
<td>6.0</td>
<td>31.9</td>
<td>2.8</td>
</tr>
<tr>
<td>Polish</td>
<td>7.0±0.1</td>
<td>20.4±0.1</td>
<td>2.3±0.2</td>
</tr>
<tr>
<td>Romanian</td>
<td>15.6±0.5</td>
<td>15.1±0.5</td>
<td>6.9±0.9</td>
</tr>
<tr>
<td>Slovak</td>
<td>14.4±0.3</td>
<td>33.7±0.3</td>
<td>4.9±1.6</td>
</tr>
<tr>
<td>Spanish</td>
<td>8.1±0.2</td>
<td>11.6±0.2</td>
<td>5.4±1.0</td>
</tr>
<tr>
<td>Turkish</td>
<td>7.0±0.1</td>
<td>25.3±0.1</td>
<td>3.7±0.2</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>14.4±0.3</td>
<td>1.7±0.3</td>
<td>1.9±0.4</td>
</tr>
<tr>
<td>×7 #samples</td>
<td>13.1</td>
<td>2.9</td>
<td>2.7</td>
</tr>
</tbody>
</table>
7.4 Training Longer

Training longer is beneficial. As can be seen in Figure 2, testing alpha-word accuracy for all our models is only rising. Under-training is most significant for Vietnamese, the language with the most diacritics. Training the corresponding model seven times longer brings substantial improvements of over 3.28%.

![Figure 2: Alpha-word accuracy improvement during diacritics restoration training.](image)

Training steps of \( \times 1 \) corresponds to 2048 \( \times 256 \) sentences for a given language. There is a visible outlier for the Turkish language at \( \times 0.5 \) training steps but that model regained the accuracy later in training.

A similar trend is observed for all the models trained on the two tasks simultaneously in Figure 3. Here the improvements are much larger. On the other hand, languages with fewer diacritics, such as French and Spanish, have diminishing gains from longer training. Overall, longer training is a must for the more difficult tasks.

Note that while the training is much longer, we still use the same dataset sizes presented in Table 1, just iterate over them more times.
8 Discussion

In this work, we show that accuracy can be improved by combining the transformer and the classical dictionary methods. Yet this is the case for more under-trained transformers. We show that longer-trained ByT5 models start to bypass the hybrid approach. However, when resources are limited compared to the difficulty of the task, such a hybrid approach can be a viable solution, as is the case with our simultaneous diacritics restoration and typos correction task.

The hybrid Dict.+ByT5 approach might have an advantage in the latter task also because the dictionary part is “trained” on the typo-free diacritization task and thus recognizes and correct typo-free words well. The ByT5 model was trained only on the combined task, it thus has a harder time learning to recognize these situations from the noisy data.

Transformer models depend on the amount of training data and small sizes can hinder the performance. Hungarian and Latvian languages, with a very similar percentage of diacritics (and hence the task difficulty), had a four-times difference between their dataset size. As a result, our achieved restoration score for Latvian was almost 2% lower. On the other hand, alpha-word accuracy of over 96% and 98% can still be reached for Latvian and Irish languages with dataset sizes of 5.5M and 1.2M words, respectively. This indicates a correlation between the difficulty of the task and the size of the dataset needed.

One way to improve our results is to leverage the fact that most of the errors are due to unseen and less-seen words in the training data. As we show in this work, longer training improves the restoration of words with moderate frequencies (0-10000) but is less effective for unseen words and is very time-consuming. The only way to improve unseen words is to rely on the additional dataset. Time constraints could additionally be relieved by employing boosting approaches [117], i.e., training on the filtered selection of data, which is known to be problematic. Such data could contain a high proportion of low-frequency and unseen words.
while at the same time being compact.

A limitation of our work is that we had only a single moderate GPU at our disposal. Scaling model size [105], incorporating additional datasets [46], and training longer can improve accuracy by several percent. Similarly, one can build a model of multiple languages to gain benefits by overlapping vocabularies and semantics of related under-represented languages, although studies report contradictory results [48, 46]. We think that all these scaling approaches are promising as future work.

In our work, we generated the typos for the entire datasets just once, but in principle, we could generate different typos each time we pass through the dataset. This would require more computation but enrich the data for longer training sessions.

Another natural future direction is the incorporation of multiple error types. It is still an active area of research as the currently achievable accuracy of such systems has a wide margin to improve [106]. In this work, we show how difficult the task becomes combining just two classes of errors. However, this is a bigger problem for classical hand-crafted approaches, but our ByT5-based ones should in principle cope with it with just additional data and training time.

9 Conclusions

We achieved 98.3% average alpha-word accuracy (within 1% of the state-of-the-art) on diacritic restoration task over 13 benchmark languages with a ByT5 universal byte-level transformer model approach, a smaller training Wikipedia dataset, and much-reduced training time (Table 3). When the training time is limited, the model is slightly improved by the assistance of a simple statistical unigram model (Dict.+ByT5). There is a solid indication, however, that longer training gets very close to the state-of-the-art even without this assistance, and still with the smaller dataset. (Figure 2).

We achieved 94.6% average alpha-word accuracy on the simultaneous diacritics restoration and typo correction task with the same models (Dict.+ByT5), training datasets and times. This is a much harder task, problematic for the specialized systems, thus we have no state-of-the-art to compare to (Table 5). There is a strong indication that longer training can significantly improve these results too (Figure 3).

We investigated that most of the errors are caused by the words that are rare in the training dataset (Table 6). However, contrary to the classical approaches, our models generalize quite well to even the unseen words (Table 7) and restore diacritics correctly on >76% of unseen words in every language. This gives us good hints on how the models can be further improved, often by simply training them more.

The good performance and universality of this approach make it very promising for real-world applications, more languages and error classes.


Funding: The research is funded by the joint Kaunas University of Technology Research and Innovation Fund and Vytautas Magnus University project “Deep-Learning-Based Automatic Lithuanian Text Editor (Lituanistas)”, Project no.: PP34/2108.
Data Availability: Publicly available datasets were analyzed in this study. This data can be found here: http://hdl.handle.net/11234/1-2607 and here: https://ufal.mff.cuni.cz/~majlis/w2c/download.html (Last accessed on 3 January 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References


