MASTER THESIS

Bc. Petr Pechman

Czech Grammar Error Correction

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Milan Straka, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2024

I would like to express my deep gratitude to my supervisor, RNDr. Milan Straka, Ph.D. and to my consultant, Mgr. Jakub Náplava, Ph.D. for the patient guidance, numerous pieces of advice and corrections, without which the thesis would not be possible. I would also like to thank my girlfriend for her support and understanding.

Title: Czech Grammar Error Correction

Author: Bc. Petr Pechman

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: We present a grammatical error correction system for correcting the Czech language. The system is based on the neural machine translation approach. We utilize the Transformer architecture, which depends on a large amount of annotated data. Given that for most languages, including Czech, there is not enough annotated data available, we opt to generate synthetic data with artificial errors. We generate not only using simple language-independent errors, but we also introduce typical Czech errors. To facilitate quick experimentation, we develop a flexible training pipeline capable of real-time data generation. Consequently, we evaluate the effect of several proposed improvements such as oversampling of language domains or a choice of data source for synthetic generation. Our best-performing model achieves state-of-the-art results in the Czech language for comparable model size. The implementation is released on GitHub at `https://github.com/petrpechman/czech_gec/tree/MasterThesis_PechmanPetr_2024`.

Keywords: Gramatical Error Correction, Synthetic Data Generation, Czech, AKCES, GECCC

Contents

Introduction

Effective communication is paramount for professional and personal success in today's digitally connected world. The ability to express oneself clearly and accurately in writing plays a critical role in achieving this communication goal. Mastering a language's grammar and syntax can pose significant challenges, especially for non-native speakers. However, achieving proficiency in these areas is crucial for effective communication, as grammar errors can impede understanding, diminish the quality of written work, and erode the author's credibility. Automatic grammatical error correction serves as a solution to address this challenge.

Grammatical error correction (GEC) is a task that aims to correct errors in texts. These errors can range from simple typos and spelling errors to more complex issues such as incorrect verb tenses, subject-verb agreement errors, punctuation errors, and syntactic inconsistencies. These errors and their corrections are usually language-specific, so development is focused on a language-specific correction system.

GEC is a long-studied task, with most research conducted in English. Due to the large availability of data, many comprehensive datasets and the best-performing systems with neural-based models have been developed for English. Other languages are also progressing in development, but not as significantly. Náplava et al. [2022] recently examined this task for Czech, created a new dataset (GECCC), analyzed metrics, and evaluated Transformer-based neural models based on it.

Our work is dedicated to a GEC system focusing on the Czech language. It extends the approach taken by Náplava et al. [2022], who addressed the low amount of annotated Czech data by generating synthetic training data using simple language-independent rules applied to clean texts.

We examine a new approach and address the main research question: *"How can the GEC system in Czech be improved by synthetically adding typical Czech errors?"*. The synthetic addition of typical errors benefits from a one-time language analysis of Czech. This analysis defines typical Czech errors, which are then introduced to the clean data. This process can generate training data that captures the Czech language much better and is readily available.

Closely related to this is the question: *"How does the performance of the GEC system depend on clean data?"*. The assumption is that clean data is completely grammatically correct and does not contain any errors, but the reality is different, and we know that even clean data can contain some grammatical errors or typos. Therefore, we evaluate several corpora with different levels of quality to assess how much the system's performance depends on the quality of the data.

Another interesting question of our work is: *"How does the model performance* *depend on its size?"*. To answer this question, we train different-sized models to evaluate how model size matters.

All three previous questions deal with improving a GEC system. We also examine optimal data generation settings, pre-training length, fine-tuning data mixing, etc., to achieve the best GEC system for the Czech language.

By carrying out this multifaceted research effort, we are trying to contribute to developing grammatical error correction systems, especially in Czech. We aim to build a state-of-the-art GEC system that effectively solves linguistic challenges specific to Czech and is a template for similar advances in other languages.

Our Contributions

All implementation and experimental work was carried out solely by the thesis author. The author's contributions are:

- **Design and implementation of a model training pipeline.** The pipeline generates real-time data and passes it directly to model training. This approach has several advantages: there is no need to generate data in advance, we can easily make changes to both the generation and training settings, and it allows execution on multiple GPUs and easy-to-run evaluations. The implementation is released on GitHub at `https://github.com/petrpechman/czech_gec/tree/MasterThesis_PechmanPetr_2024`.
- **Analysis of typical Czech errors and their addition to the automatic generation of noisy data.** Typical Czech errors significantly improve the performance of our GEC system.
- **Analysis of the data source for the GEC system using typical Czech errors.** We compare different corpora for model pre-training with different cleanliness levels and find that the cleaner the corpus is, the better the system performs.
- **Domain oversampling.** We try an approach of oversampling individual domains in different ways when training the model, achieving better performance on the GECCC dataset [Náplava et al., 2022].
- **Comparison of different model sizes.** Larger models achieve better results.
- **Evaluation of pre-trained models.** Our model is the original Transformer [Vaswani et al., 2017]. We experiment with replacing it with a pre-trained MT5 [Xue et al., 2021] and compare the results.
- **The state-of-the-art GEC system in Czech for comparable model size.** Our final GEC system achieves state-of-the-art results for the Czech language for comparable model size.

1. Background

1.1 Evaluation

The evaluation of GEC systems is based on edits. An edit is the correction of the subsequence of the source sentence. It is specified by the start and end in the source sentence and the text to be inserted at that part of the sentence to remove the error. Edits have the advantage that they can contain an entire error, which can span multiple words.

Example of edit in M2 format:

```
S Co je pro mně důležité ?
A 3 4|||MeMne|||mě|||REQUIRED|||-NONE-|||0
```

Edits also have another advantage, which may not be obvious, and that is the inverse operation, i.e., we can introduce an error into the correct sentence. For example, we use this in our approach of adding typical Czech errors.

Edits are divided into system and gold. System edits are corrections generated by the GEC system, and gold edits are corrections from annotators, where we assume their correctness.

Edit evaluation is done by computing Precision, Recall, and F$_1$-score [Blair, 1979] between the set of system edits $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ and the set of gold edits $\{\mathbf{g}_1, \ldots, \mathbf{g}_n\}$ for all sentences

$$P = \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{e}_i|}, \tag{1.1}$$

$$R = \frac{\sum_{i=1}^{n} |\mathbf{e}_i \cap \mathbf{g}_i|}{\sum_{i=1}^{n} |\mathbf{g}_i|}, \tag{1.2}$$

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}, \tag{1.3}$$

where we define the intersection between $\mathbf{e}_i$ and $\mathbf{g}_i$ as [Dahlmeier and Ng, 2012]:

$$\mathbf{e}_i \cap \mathbf{g}_i = \{e \in \mathbf{e}_i | \exists g \in \mathbf{g}_i (match(e, g))\}. \tag{1.4}$$

F$_{0.5}$-score is introduced to better capture the performance of the GEC system because the F$_{0.5}$-score emphasizes Precision more than Recall. This metric correlated better with human judgment because leaving an uncorrected error is not as bad as using the wrong edit [Ng et al., 2014].

$$F_{0.5} = \frac{(1 + 0.5^2) \cdot P \cdot R}{0.5^2 \cdot P + R}, \tag{1.5}$$

An essential part of the evaluation is extracting edits. The GEC system usually does not generate edits, but straightaway corrects sentences. We need to get edits from the original uncorrected sentence (system hypothesis) and the newly generated corrected sentence (source sentence) for evaluation.

Dahlmeier and Ng [2012] present evaluation tool MaxMatch scorer finding edits between a source sentence and a system hypothesis that achieves the highest overlap with the gold-standard annotation. These optimal edits are then scored using the F-score. They use the Levenshtein distance [Levenshtein, 1966] between a source sentence and a hypothesis computed in the Levenshtein matrix. Each vertex in the graph corresponds to a cell in the matrix, and each edge corresponds to an atomic edit operation: inserting, deleting, substituting, or leaving a token unchanged. Each path corresponds to the shortest sequence of atomic edit operations that transform the source sentence into a hypothesis. They assign a unit cost to each edge in the lattice.

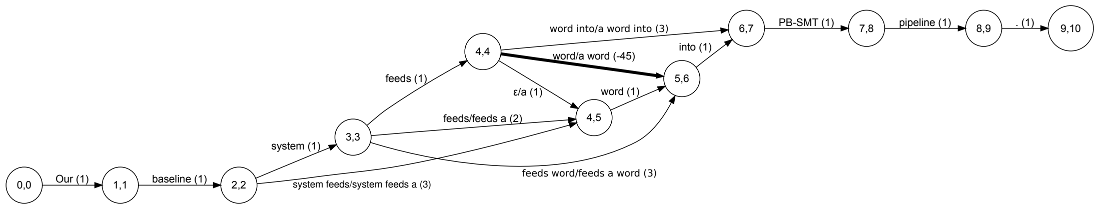

Figure 1.1: The edit lattice for “Our baseline system feeds ($\epsilon \rightarrow$ a) word into PB-SMT pipeline” Edge costs are shown in parentheses. The edge from (4,4) to (5,6) matches the gold annotation and carries a negative cost. [Dahlmeier and Ng, 2012]

To allow for multi-token edits, they add transitive edges to the graph as long as the number of unchanged words in the newly added edit is not greater than the limit (u) and the edit changes at least one word. The cost of a transitive edge is the sum of the costs of its parts. The graph extracted from the example sentence is shown in Figure 1.1 taken from Dahlmeier and Ng [2012]. They change the cost of each edge whose corresponding edit has a match in the gold standard to $-(u + 1) \times |E|$.

Then, they perform a single-source shortest path search with negative edge weights, which can be done with the Bellman-Ford algorithm [Cormen et al., 2001].

Another well-known evaluation tool is ERRANT [Bryant et al., 2017]. Bryant et al. [2017] extract edits using a linguistically enhanced alignment algorithm supported by a set of merging rules from Felice et al. [2016]. Further, tokens of source and system sentences are POS tagged, lemmatized, stemmed, and dependency parsed. Due to the extraction of edits and token tagging, the rule-based classifier can classify edits and evaluate a specific type of error. For example, you can choose only *replacement* errors or more specific *replacement noun* errors.

1.2 Datasets

Datasets are a necessary part of GEC, a source of quality data, and therefore they allow us to build and evaluate our GEC systems. These datasets are usually manually annotated by experts for specific languages. There are many datasets for different languages, most of them for English.

The Lang-8 corpus [Tajiri et al., 2012] is one of them, created from texts written by English-as-a-second-language (ESL) learners and corrected by native speakers. It consists of approximately 120,000 English entries containing 2,000,000 verb phrases, with 750,000 verb phrases having corrections.

Another English corpus is NUCLE [Dahlmeier et al., 2013], which is created from student essays written by undergraduate students at NUS. These essays were annotated by CELC English instructors. In total, this corpus has 46,597 error tags.

The FCE [Yannakoudakis et al., 2011] corpus is composed of scripts written by learners taking the First Certificate in English (FCE) exam. Each script has between 200 and 400 words and is manually tagged with information about the linguistic errors committed.

There are two main datasets for the Czech language: AKCES-GEC [Náplava and Straka, 2019] and GECCC [Náplava et al., 2022].

1.2.1 AKCES-GEC

Náplava and Straka [2019] present the AKCES-GEC dataset, composed of resources CzeSL (learner corpus of Czech as a second language, originally created by Rosen [2016]) and ROMi (Romani ethnolect of Czech Romani children and teenagers) created within the AKCES project [Šebesta, 2010], containing additional resources: SKRIPT and SCHOLA – written and spoken language collected from native Czech pupils. The dataset is split into train/dev/test parts. Development and test parts are annotated by two annotators to enable better evaluation. All parts are saved in the M2 format [Dahlmeier and Ng, 2012].

1.2.2 GECCC

The Grammar Error Correction Corpus for Czech (GECCC) [Náplava et al., 2022] is a large and diverse dataset containing 83,058 sentences from various domains. It is one of the largest GEC datasets and it is composed of several datasets or their subsets: SKRIPT 2012 (AKCES project), Facebook Dataset

[Habernal et al., 2013], Czech news site novinky.cz, the ROMi corpus (AKCES project), the ROMi section of the AKCES-GEC corpus, the Foreigners section of the AKCES-GEC corpus, and the MERLIN corpus [Boyd et al., 2014]. Therefore, GECCC covers four main domains: essays by native Czech pupils, informal web discussions written primarily by Czechs, essays by Roma children and teenagers, and non-native student essays. Each domain has a different representation size; see Table 1.1. The dataset is divided into train/dev/test parts and stored in the M2 format [Dahlmeier and Ng, 2012] and in a plain-text format.

Domain	Train	Dev	Test	Error Rate
Natives Formal	4 060	1 952	1 684	5.81%
Natives Web Informal	6 977	2 465	2 166	15.61%
Romani	24 824	1 254	1 260	26.21%
Second Learners	30 812	2 807	2 797	25.16%
Total	66 673	8 478	7 907	18.19%

Table 1.1: The number of sentences in the GECCC dataset's domains. The average error rate was computed by concatenating development and test data.

1.3 GEC Systems

We now describe several grammar error correction systems. They are based on different approaches, some of them using rules, others using statistical methods, some using Encoder from the Transformer architecture, and others using the complete Transformer architecture. Transformer-based systems (specifically the one described in Section 1.3.3) are closely related to our approach of Czech GEC. We also present a few systems dealing with only specific phenomena instead of a full-scale grammar error correction.

1.3.1 Korektor

A forerunner of today's systems, Korektor [Richter et al., 2012] is a tool for Czech text correction. It uses a statistical model based on the hidden Markov Model to suggest word variants (hidden states). Language n-gram models specify transition costs and the Viterbi algorithm determines the passage through the hidden states. This tool performs well in correcting typos and diacritics, but it is limited to correcting words only, for example, it cannot add a word.

1.3.2 Opravidlo

Opravidlo is a complex GEC tool for Czech and its development involved an analysis of the most common errors in Czech. It is a rule-based system using five different rule modules targeting a specific range of errors. It also uses a spell-checker with an extensive dictionary to correct typos [Hlaváčková et al., 2022].

1.3.3 Encoder-Decoder Systems

Nowadays, neuron-based models built on encoder-decoder architecture achieve state-of-the-art grammatical error correction performance, especially for morphologically rich languages such as Czech. The approach of Encoder-Decoder models is the most direct of all mentioned. Its input is a noisy sentence, and its output is a corrected sentence; it works similarly to translation.

Brockett et al. [2006] are the first to propose this idea, in which the correction of grammatical errors is solved as a translation from noisy incorrect sentences into corrected sentences. This approach allows the correction task to be treated as SMT (Statistical Machine Translation) problem that they solve using the noisy channel model [Brown et al., 1993].

Yuan and Briscoe [2016] are the first to use this translation approach for NTM (Neural Machine Translation) based on the RNNsearch model with a bidirectional RNN as an encoder and an attention-based decoder [Bahdanau et al., 2016].

Grundkiewicz et al. [2019] use the same approach with NMT, but replace the RNNsearch model with deeper neural networks, where the Transformer model [Vaswani et al., 2017] achieves the best results.

Synthetic-trained Transformer

Náplava and Straka [2019] use a GEC system based on the Transformer model [Vaswani et al., 2017], which is improved by techniques such as source and target word dropouts, edit-weighted MLE, and checkpoint averaging. The pre-training of the model is based on Grundkiewicz et al. [2019] using an unsupervised approach to create noisy input sentences, it is more described in Section 1.4.2.

Fine-tuned Transformer

The model described by Náplava and Straka [2019] follows the previous Transformer approach described above, it takes the pre-trained model on synthetic data and fine-tunes it on a mixture of the AKCES-GEC dataset and synthetic data. This mixture avoids early overfitting and improves model performance significantly.

Náplava et al. [2022] use also the same pre-trained model and fine-tune it on a mixture of synthetic data and the GECCC dataset. For the Czech language, a mixture of 2:1 is used, and it has a better $F_{0.5}$-score than a model fine-tuned only on a mix of AKCES-GEC and synthetic data. The evaluation of both models is performed on the GECCC test set.

1.3.4 Non-Autoregressive GEC Tagging

The encoder-decoder architectures are computationally demanding and slow during inference. To address these limitations, recent approaches propose token-tagging methods, which achieve significant speedups.

Omelianchuk et al. [2020] introduce English GEC sequence tagger using a Transformer encoder. This approach transforms GEC into a sequence tagging problem that benefits from the Transformer architecture and speed of the tagging system. The sequence tagger is pre-trained on synthetic data and then fine-tuned in two stages: first on errorful corpora and then on a combination of errorful and error-free parallel corpora. Developing custom token-level (word-level) transformations increases the coverage of grammatical error corrections, and iterative usage of the sequence tagger improves output quality.

A similar approach is taken by Straka et al. [2021], who created a Czech character-based non-autoregressive GEC system with automatically generated character transformations. The use of token-level taggers would bring a large increase in rules for morphologically rich languages. Therefore, they choose character-level transformations, which address errors like spelling errors, diacritics, and morphology. By automatically inferring transformations from the corpus, they avoid the need for manual rule design and achieve efficient handling of character-level corrections. The results show good performance but do not reach the level of the Encoder-Decoder architecture. The effectiveness of non-autoregressive systems decreases mainly for morphologically rich languages (e.g. Czech). However, it brings significant speed-up compared to autoregressive systems.

1.3.5 Systems Focused on the Specific Error Type

Here we list a few systems that deal with specific error types. These systems can be significantly simpler than a complex GEC system. Some of them are or may become part of complex GEC systems.

RNN-based Correction Diacritics

Náplava et al. [2018] describe a model architecture focused on adding diacritics to input characters. Diacritics are orthographic marks added to letters that indicate variations in meaning or pronunciation. The model uses a bidirectional recurrent neural network (RNN) [Graves and Schmidhuber, 2005] to process input characters. In addition, during inference, the model uses a left-to-right beam search decoder combining the probabilities from the output of the RNN and a language model. This integration of RNN and a language model improves the decoding process and enables better character diacritics. Among other things, the system is trained and evaluated on Czech diacritics.

Punctuation Correction Based on SET

Kovář [2014] presents an automatic method for correcting errors in commas and subject-predicate agreement. The system is based on a rule-based syntactic analysis provided by the SET parsing system [Kovář et al., 2011]. The system has ten rules and emphasizes precision in error detection to minimize false alerts. Kovář et al. [2016] extend this system by adding new rules and obtaining significantly higher recall but slightly lower accuracy.

CzAccent

The CzAccent system [Rychlý, 2012] operates on the principle of utilizing a large lexicon for Czech words to restore accents in text. For every word, the most frequently accented word from all possible accented words and the original non-accented word is selected and added to the CzAccent lexicon. Then, it uses a straightforward approach involving reading words, searching for them in the lexicon, and printing the accented variant if found or the original word if not. This simplicity contributes to the system's speed, allowing it to quickly process a large amount of data.

BERT-based Correction Diacritics

Náplava et al. [2021] introduce a BERT-based approach to the task of correcting diacritics in the given text and achieve significant improvements in Czech. The model architecture uses a pre-trained multilingual BERT model [Devlin et al., 2019] to obtain contextualized embedding processed through a fully-connected feed-forward neural network producing probabilities of diacritics instructions for sub-words.

Classifier of Homophones

The classification of homophones task [Švec et al., 2020] is focused on correcting homophones y/i in the Czech language. It differs from all other systems because its input is output from an ASR decoder. Although its main goal is a bit different, the approach to solving the problem is very similar to Transformer-based systems focused on GEC. Because this system is aimed at solving one particular task (a decision between i and y), a classifier can be used, specifically a Transformer with the BERT architecture [Devlin et al., 2019].

1.4 Methods for Data Generation

While a machine translation–based approach to grammar error correction delivers superior results, especially when employing the Transformer-based encoder-decoder architecture, such a system requires a vast amount of data on the order of billions of words. Annotated corpora by language experts are an ideal data source, but they are insufficient for most languages. Therefore, automated methods are needed to generate the required amount of data.

1.4.1 Wikipedia Corpus Extraction

The first described approach for generating synthetic errorful sentences is to extract parallel sentences from Wikipedia revision history. This way, a large corpus can be obtained, especially for English. However, since Wikipedia edits are not created specifically for GEC purposes, the resulting corpus tends to be highly noisy. An example of a corpus created by extraction from Wikipedia is Grundkiewicz and Junczys-Dowmunt [2014]

1.4.2 Synthetic Corpus Generation

Another popular approach is to generate a synthetic corpus from a clean monolingual corpus using data noising.

Grundkiewicz et al. [2019] present the synthetic generation of noisy sentences from error-free sentences. For each sentence, they sample probability p_{err} from a normal distribution resembling the word error rate of the development set. This is multiplied by sentence length and rounded to a number of words to change. We apply one operation (deletion, substitution, swapping with a neighbor, and inserting a random word) for every chosen word with a given probability. The probability for word substitution is 0.7, and the three other operations are chosen with a probability of 0.1. The same method to obtain noisy data is applied to individual characters in 10 percent of words.

This approach is also used by Náplava and Straka [2019]. They introduce errors to synthetic data for Czech, English, German, and Russian. After analyzing the results on texts, they added another word-level edit to change capitalization and a char-level edit to change diacritics for Czech.

Another approach is used by Yuan and Felice [2013], they extract correction patterns from the NUCLE corpus. They then rewrite these correction patterns on the error generation patterns, which they then apply on the clean data.

Corpora

We present several clean corpora that we also use to generate synthetic noisy texts in our work (especially in Section 3.1 and Section 3.5).

- The SYN-v4 corpus [Křen et al., 2016] is a representative collection of recent written Czech, containing over 3.5 billion word tokens. It contains all synchronous written corpora of the SYN series (SYN2000, SYN2005, SYN2006PUB, SYN2009PUB, SYN2010, SYN2013PUB, SYN2015) and, in addition, it includes journalism mainly from 2010-2014.
- The News 2019 corpus [Barrault et al., 2019] is a monolingual Czech corpus created together with others for the WMT 2019 news translation shared task.
- The Wikipedia corpus [Kubeša and Straka, 2023], presented within DaMuEL, is a large Multilingual Dataset for Entity Linking containing data in 53 languages. We use only raw sentences extracted from Wikipedia, which makes up our Wikipedia corpus.
- The Common Crawl corpus [Ginter et al., 2017] is created from downloaded texts from the internet and was created for the CoNLL 2017 Shared Task.

Table 1.2 shows us the sizes of the corpora. We use subsets of corpora because they are otherwise too large. Each subset consists of individual sentences; each sentence is on a separate line.

Corpus	Num of Sentences
SYN-v4	28 619 909
News 2019	14 645 268
Wikipedia	9 743 253
Common Crawl	75 594 393

Table 1.2: Corpora sizes.

2. Description of Our System

In this thesis, we build a parametrized pipeline composed of two main parts, the transformer-based model and the data generation. Data generation happens simultaneously during model training and fills batches in real-time. This avoids the one-time creation of a synthetic corpus, which could limit long-time pre-trainings that need a large amount of data, and also allows us to easily change the automatic text noising and model settings.

2.1 Model

One of the best solutions in the field of GEC in Czech is the GECCC fine-tuned Transformer presented by Náplava et al. [2022] (Section 1.3.3). We decided to follow this approach because Transformer-based architecture is the state-of-the-art approach for GEC.

The Transformer architecture [Vaswani et al., 2017] contains an encoder for processing the input and a decoder for generating the output. Both components utilize self-attention mechanisms, allowing the model to capture contextual dependencies effectively. The biggest benefit of this architecture is enabling parallel processing and efficiently capturing long-distance dependencies.

Our pipeline is built in such a way that it is possible to change the model, the only requirement is any encoder-decoder model supporting sequence-to-sequence and a corresponding tokenizer. Following Náplava et al. [2022], we choose the same model named "Transformer", defined in the Tensor2Tensor[1] library. The Tensor2Tensor is no longer being developed, so we use the TensorFlow[2] framework, where we define the "Transformer" model and use this model in base size whose hyper-parameters are described in Table 2.1.

Model	Dim	Layers	Heads	FFN	Params
base	512	6	8	2 048	65×10^6

Table 2.1: Hyper-parameters of the Transformer model.

2.2 Data Generation

To train a sequence-to-sequence model based on Transformer, it is necessary to have enough data that contain input sequence and sample output sequence, especially pairs of incorrect-correct sentences. The best source of data is a corpus

[1] `http://github.com/tensorflow/tensor2tensor`

[2] `https://www.tensorflow.org/`

annotated by human language specialists. Unfortunately, there is a lack of such data for Czech, or rather it is a problem for almost every language. Therefore, we have to create such data automatically. There are many possible approaches to solving the lack of data (described in Section 1.4).

We choose the approach of generating synthetic corpora. The synthetic corpus is generated during model training, not beforehand. We also utilize already-known methods and newly implemented the generation of typical errors for the Czech language. Data generation can be divided into two parts: artificial errors and typical errors. These parts are connected; artificial errors are introduced first, and then typical errors are generated.

2.2.1 Artificial Errors

We generate synthetic data from a clean monolingual corpus (Section 1.4.2). We follow Náplava and Straka [2019] and Grundkiewicz et al. [2019] using an unsupervised approach to create noisy input sentences. For each correct sentence, we sample a probability, *perr_word*, from a normal distribution with a preset mean and standard deviation. This probability is then multiplied by the number of words in the sentence, and it determines how many words will be modified. Each selected word is modified by one of several predefined operations with specific probabilities: deletion, swapping with its adjacent right neighbor, insertion of a random word from the dictionary after the current word, or substitution with one of its suggestions (Aspell/MorphoDiTa). It is shown in detail in Figure 2.1.

Náplava and Straka [2019] state that for better robustness of the system to spelling errors, it is a good idea to use the same operations for individual characters. Additionally, they extend word-level operations to contain the change of the word casing. If a word is chosen for case changing, a word is converted to lower-case with 50% probability, or some individual characters are chosen and their casing is changed. Furthermore, they add a new character-level operation that generates one of the diacritical variants or removes the diacritics for the selected character.

The token-level substitution is greatly influenced by how the suggestions are selected. Suggestions can be selected from a dictionary or a lexical network. Náplava and Straka [2019] utilize suggestions by a spell-checking tool Aspell.

```
Input:
    sentence: Viděl jsem v lese velkého medvěda .

mean_token, std_token = 0.15, 0.2
mean_char, std_char = 0.02, 0.01
token_operation_probs: [substitution: 0.7,
    insert: 0.1,  delete: 0.05,
    swap: 0.1, change_casing: 0.05]
char_operation_probs: [substitution: 0.2,
    insert: 0.2, delete: 0.2, swap: 0.2,
    change_diacritics: 0.2]

p_err_token = sample_normal_dist(mean_token, std_token)
count_of_word = count_words(sentence)
count_of_changes = round(p_err_token * count_of_word)

for i in 1..count_of_changes:
    index = randomly_select_word_index()
    word = get_word(sentence, index)
    operation = select_operation(token_operation_probs)
    word = apply_operation(word, operation)
    sentence = replace_word(index, sentence, word)

p_err_char = sample_normal_dist(mean_char, std_char)
count_of_chars = count_chars(sentence)
count_of_changes = round(p_err_char * count_of_chars)

for i in 1..count_of_changes:
    index = randomly_select_char_index()
    char = get_char(sentence, index)
    operation = select_operation(char_operation_probs)
    char = apply_operation(char, operation)
    sentence = replace_char(index, sentence, char)

return sentence
```

Figure 2.1: Generating a noisy sentence from an error-free sentence by simple language-independent error introduction.

Aspell

Aspell proposes candidate words that are lexically and phonetically similar but also valid words, thereby introducing errors that are hard to detect and correct [Náplava, 2022]. For example, it can generate errors in subject-verb agreement, verb tense, or morphology (see example on Figure 2.2).

```
Aspell:
    input: medvěda
    output: ['medvěda', 'Nedvěda' 'medvěd', 'medvěde',
            'medvědi', 'medvědu', 'medvědy', 'medvědě',
            'medvědí', 'medvědů', 'med věda', 'med-věda']
```

Figure 2.2: Example of possible substitution words generated by Aspell.

MorphoDiTa

We also add the MorphoDiTa [Straková et al., 2014], which allows us to find related words that are morphologically similar or derived from each other (see example on Figure 2.3). Morphological Dictionary and Tagger (MorphoDiTa) is an open-source tool that offers advanced morphological analysis, morphological generation, tagging, and tokenization for the Czech language. It incorporates MorfFlex CZ 2.0 [Hajič et al., 2020] – a morphological dictionary used for analyzing and generating Czech word forms, and DeriNet data [Vidra et al., 2021] – a lexical network storing derivational relations among Czech lemmas. By merging these resources, MorphoDiTa provides a comprehensive tool for exploring morphological phenomena in Czech, enabling users to traverse derivational trees and access detailed linguistic information efficiently [Žabokrtský et al., 2016].

```
MorphoDiTa:
    input: medvěda
    derinet_distance: 2
    output: ['medvědáriu', 'nemedvídkovštějších',
            'nemedvědovitejch', 'nejmedvídkovitějším',
            'medvědice', 'medvíďátku',
            'medvídkářštějšími', 'nemedvědovitostech',
            'nemedvídkovskýma', 'méďovými', ...]
    output_length: 684
```

Figure 2.3: Example of possible substitution words generated by MorphoDiTa.

2.2.2 Typical Errors

The amount of typical Czech errors in synthetically generated noisy texts is relatively small because typical errors are too complex for approaches such as the Synthetic Corpus, and their representation in the Wikipedia Extracting Corpus is not guaranteed. In corpora, AKCES-GEC (Section 1.2.1) and GECCC (Section 1.2.2), some of these errors are represented, but unfortunately, datasets are too small for robust pre-training. By adding specific errors, we assume an overall improvement of the GEC system. We would also like to analyze how the model behaves on specific typical Czech errors.

In our approach, after introducing artificial errors, we add typical errors for Czech regardless of how the words/characters were changed in the previous part. In Section 2.3, we analyze typical errors for the Czech language, from which we obtain the basic typical errors for the Czech language. An error is defined as an edit, i.e., where in the sentence the error begins, where the error ends, and what the given sub-sequence should be replaced with M2 format (see the M2 format described in Section 1.1).

We do not introduce a typical error into the data whenever we can. We need to train the model on both noisy and clean sentences. Therefore, for each typical error, we have a defined probability with which the error is introduced into the data. This probability can be absolute or relative.

If we can introduce the given error into a sentence, we decide with absolute probability whether to apply it or not.

On the other hand, the relative probability expresses the probability of an error on any token. For example, we have 20 sentences, each with 10 tokens. Error A has a relative probability of 0.02, so we want to introduce error A four times because $20 \cdot 10 \cdot 0.02 = 4$. If we have multi-word errors, we count them as single-word errors. For example, we have a sentence with ten tokens, and error B affects two words. If we introduce error B into the sentence, the relative probability of error B on the sentence is $\frac{1}{10} = 0.1$.

For each typical error, only one type of probability can be used to generate data. We go through the data sentence by sentence and generate all possible errors for each one. Then, if two sentence errors overlap, we randomly pick one and remove it. We repeat this process until only non-overlapping errors remain. We thus obtain a set of feasible errors, which we introduce into the sentence according to the given probability (absolute or relative).

2.3 Analysis of Czech Typical Errors

We have identified common grammatical errors in Czech that frequently appear in both written and spoken language. We aimed to address a wide range of these errors to improve the entire GEC system.

The first typical error is the forms of the words “mě” and “mně”, which are often confused. These forms of the word “já” are often confused because they are pronounced the same. Example: correct “Přišel ke mně.” vs. incorrect “Přišel ke mě.”, translated: “He came to me.” Both sentences have the same pronunciation, but using the third case determines the form “mně” [Plocková, 2019]. The parts of the words “mě” and “mně” are also often interchanged in the middle or at the end of words. For example, in the word “upřímně”, the suffix “-mně” is used because this word is derived from the word “upřímný” [Plocková, 2019]. In the same way, parts of the words “bě” and “bje” can be interchanged (Example: “objet” vs. “obět”).

The next common mistake involves improper capitalization, which is governed by numerous rules [Vostřelová, 2019]. Capital letters are written, for example, in personal names, at the beginning of sentences, in official names of geographical places, etc.

Another crucial area of error is punctuation. Punctuation marks are used for text segmentation and meaning clarification. People often make punctuation errors due to a lack of knowledge about punctuation rules or simply overlooking the importance of proper punctuation [Machura, 2022].

Next, we examine typical errors for second-grade elementary school students [Šmatová, 2015]. Here are examples of errors:

- Typing “i” or “y” (same for “í”/“ý”) after specific letters (“b”, “f”, “l”, “m”, “p”, “s”, “v”, “z”). For Czech, there is a set of words in which these selected letters are followed by “i” or “y”. The letter (“i”/“y”) is fixed for each such word. Example: “mlýn”.

- Interchanging the prepositions “s” and “z” due to the same pronunciation. The spelling follows the case of the preposition. In connection with the second case, we write “z”, and in connection with the seventh case, we write “s”. Example: “ze skříně” – means from the closet, “se skříně” – means to get something away that was on the closet.

- Interchanging the prefixes “s-” and “z-” – errors occur here due to the similarity of pronunciation. Some words have only one correct prefix, but there are words where both prefixes are possible. For those with a change in the prefix, there is also a change in meaning.

- Interchanging the letters “ů” and “ú”, which are pronounced the same. Writing “ů”/“ú” follows a simple rule that “ú” is written at the beginning of a word and “ů” is written in the middle or at the end of a word, but there are many exceptions.

- Interchange of endings “-i”/“-í” and “-y”/“-ý” for nouns, adjectives, and verbs. This is one of the most common mistakes. To write “i”/“y” (“í”/“ý”) correctly, one must know Czech grammar and phenomena such as the agreement of the subject with the predicate, types of adjectives, etc.

Other typical errors that can appear in Czech and are worth mentioning are errors in diacritics. The Czech language uses 42 letters. Of these, 27 are from the basic Latin alphabet, and 15 have three different types of diacritical marks (“ˇ”, “´”, “°”). Diacritical marks distinguish different meanings of the same letter.

Here, we also have errors, such as incorrect forms of conditionals. For example, the correct “My bychom vyhráli.” and the incorrect “My bysme vyhráli”, translated: “We would win.”

Errors are also often made in the use of written numerals (“dvěma/dvouma”), the word “sebou”, which is sometimes used with the preposition “s”, or specific words such as “výjimka”, which is confused with “vyjímka” or “viz”, which is written without a period.

From this analysis of typical errors, we create a list of errors we introduce into the data. Some errors in the list correspond precisely to the analysis, some cover several errors at once, and others, on the contrary, cover only a part. Table 2.2 contains all the errors we generate, including examples of clean and noisy sentences with marked changes.

Name	Example – Clean	Example – Noisy
word mě/mně	Přišel ke **mně**.	Přišel ke **mě**.
suffix -mě/-mně	Ohro**mně** se bavil.	Ohro**mě** se bavil.
infix -mě-/-mně-	On je rozu**mně**jší.	On je rozu**mě**jší.
suffix -i/-y	Kluci jel**i** domů.	Kluci jel**y** domů.
letter d/t/n followed by -i/-y	Mlad**ý** muž.	Mlad**í** muž.
letter b/f/l/m/p/s/v/z followed by -i/-y	Ob**y**vatelé města.	Ob**i**vatelé města.
letter ů/ú	Jdu dom**ů**.	Jdu dom**ú**.
conditionals	Byli **bychom** rádi.	Byli **bysme** rádi.
specific words	To je **výjimka**.	To je **vyjímka**.
prefix s-/z- or se-/ze-	On **s**hrabal listí.	On **z**hrabal listí.
word forms denoting count	Jeli **oběma** auty.	Jeli **oběmi** auty.
word mi/my	Dej **mi** knihu.	Dej **my** knihu.
suffix -bě/-bje	Našel v so**bě** odvahu.	Našel v so**bje** odvahu.
prefix bě-/bje- & infix -bě-/-bje-	Co je k o**bě**du?	Co je k o**bje**du?
phrase sebou/s sebou	Přines to **s sebou**.	Přines to **sebou**.
The first character in the sentence:		
– uppercase to lowercase	**P**ostalvil dům.	**p**ostalvil dům.
– lowercase to uppercase	**t**oto je poznámka	**T**oto je poznámka
The first character in any word:		
– uppercase to lowercase	Viděl jsem **V**aška.	Viděl jsem **v**aška.
– lowercase to uppercase	Krásné **m**ěsto.	Krásné **M**ěsto.
preposition s/z	Volby budou kdo **s** koho.	Volby budou kdo **z** koho.
adding a comma	Hlavní město má historické a krásné centrum.	Hlavní město má historické**,** a krásné centrum.
removing a comma	Navštívil město**,** kde vyrůstal.	Navštívil město kde vyrůstal.
adding diacritics	Nic ho nen**a**padlo.	Nic ho nen**á**padlo.
removing diacritics	On mi zavol**á**.	On mi zavol**a**.

Table 2.2: Typical Errors.

2.4 Technical Solution of the Pipeline

We are coming up with a pipeline implementation that improves and simplifies the overall process of model training and data generation. Usually, these two processes are separated, where the researcher first generates the data and then trains the model on that data.

We have combined these processes into a single pipeline so that we generate real-time data that is immediately used to train the model.

To connect these processes, we use the tools provided by the TensorFlow library, which allow to read data, transform it, and pass it to batch processing, after which the batches are passed to training.

The first problem was the slow reading of data from one process; we, therefore, had to parallelize this reading. We allow reading iteratively across multiple files. For every file, we run multiple reading processes, each assigned a part of the file.

Another problem was the Aspell tool, which is time-consuming and allocates more and more memory due to a memory leak. We solved the memory issue[3] and overcame the low Aspell runtime performance with parallelization. By already reading the data in parallel, we added to the reading the process of generating synthetic errors, both artificial and typical.

The next problem was data batching. Usually, the data is bundled into a batch, and if the individual samples have different lengths, they are padded. This approach was suboptimal in our case because we have different sentence lengths that can be very different, so we employed batching by buckets[4]. This means we bundle similar length samples into a bucket and pad them by just a few tokens. This approach allows us to limit the ratio of the padding tokens used in every batch, resulting in nearly optimal effective batch sizes.

Another benefit that our pipeline brings is easy configurability. If we run an experiment, everything is set up in one configuration file, which contains information about what model to use, what data, how the data should be noised, which optimizer and how it should be set, also contains the batch sizes, the paths to the evaluation datasets, etc.

We also split the evaluation into two parts: the first part is the prediction, and the second is the evaluation itself. Such a division is motivated by the fact that the prediction requires a GPU card, while the evaluation is handled by the CPUs. So, in this way, we optimize the use of graphics cards.

We also enable training (and inference) on multiple graphics cards using TensorFlow and MirroredStrategy[5].

[3]`https://github.com/WojciechMula/aspell-python/issues/23`

[4]`https://www.tensorflow.org/api_docs/python/tf/data/Dataset#bucket_by_sequence_length`

[5]`https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy`

3. Experiments

By default, model training is divided into two stages: pre-training and fine-tuning. The pre-training stage is computationally demanding and time-consuming. During this phase, the model learns general language patterns, syntactic structures, semantics, and mainly grammatical corrections. This stage corresponds to the situation where we have no annotated data but want to build a model for grammar correction. The fine-tuning stage is less computationally demanding and aims to improve the model in grammar corrections further. We perform it on annotated data in the Czech language. We also often use the approach of mixing synthetic and annotated data to avoid model overfitting.

Unless otherwise noted, all graphs we present are evaluated on the development data to show training progress; conversely, the results shown in the tables are evaluated on the test data, where we are already making the final comparison of runs. For each run, we select a checkpoint according to the highest score on the development data and evaluate it on the test data using the $F_{0.5}$-score. We perform evaluations using the MaxMatch (M^2) scorer (Section 1.1).

Overview of Experiments

In Section 3.1, we first focus on the pre-training stage, where we compare different settings for generating synthetic noisy data. Then we turn to Section 3.2, where we examine the dependence of the fine-tuning stage on the length of the pre-training stage. Then, we try to find the best way of fine-tuning for the AKCES dataset (Section 3.3) and, subsequently, for the GECCC dataset (Section 3.4), where we also analyze the use of domain oversampling. In Section 3.5 we deal with the quality of different corpora and how their quality affects the performance of the model. In Section 3.6, we enlarge the model to get the best results and also train a smaller model for comparison. We also examine whether using an MT5 model, i.e., a model pre-trained on a variety of NLP tasks instead of a randomly initialized model for pre-training, leads to a better GEC model.

Model Configuration

Unless otherwise stated, we train on a single GPU (NVIDIA RTX A4000); for pre-training, we use the AdamW optimizer [Loshchilov and Hutter, 2019] with a learning rate of 5e-05 and a weight decay of 0.01. The batch size is 128, the epoch length is 128 000 steps ($128 \cdot 128\,000 = 16\,384\,000$ samples per epoch), and the maximal length of a sample is 128 tokens.

We use the same AdamW optimizer also for fine-tuning, but we lower the learning rate to 5e-6. The batch size is still 128, and the epoch length is 330 steps for the AKCES dataset and 938 for the GECCC dataset. The pre-training time of one epoch is approximately 12 hours. The fine-tuning time of one epoch is from 4 minutes to 12 minutes, depending on the data size.

In the evaluation, we decode using a beam search of size 4.

3.1 Synthetic Data Generation for Model Pre-training

In the pre-training stage, the quality of the synthetic data plays an important role. We try four ways of data noising that are preceded by the same generation of artificial errors (Section 2.2.1) with different parameter settings. Then, we optionally add typical errors for the Czech language. Setting the parameters is based on experiments by Náplava and Straka [2019].

We have named each setting according to a specific parameter or method (when we write about settings, we write in italics – e.g., a setting that uses only Aspell for substitution and no typical errors is denoted as *Aspell*). The **MATE** setting is a combination of all the others, so it comes from names ***M**orphoDiTa*, ***A**spell* and ***T**ypical **E**rrors*.

For all settings, we have the same probabilities of character-level operations (see Table 3.1). The settings differ only in token-level substitution (see Table 3.2), where there are two tools to choose from: Aspell (see Section 2.2.1) and MorphoDiTa (see Section 2.2.1). Typical errors are introduced into data only in *Typical Errors* and *MATE*.

Char-level Operations				
sub	ins	del	swap	diacritics
0.2	0.2	0.2	0.2	0.2

Table 3.1: Probabilities for character-level noising operations.

	Token-level Operations						
Setting	sub Aspell	sub MorphoDiTa	ins	del	swap	recase	typical errors
Aspell	0.7	0.0	0.1	0.05	0.1	0.05	No
MorphoDiTa	0.5	0.2	0.1	0.05	0.1	0.05	No
Typical Errors	0.7	0.0	0.1	0.05	0.1	0.05	Yes
MATE	0.5	0.2	0.1	0.05	0.1	0.05	Yes

Table 3.2: Probabilities for token-level noising operations and using of typical errors.

All settings use the same clean data – the SYN-v4 corpus to generate noisy data.

As Figure 3.1 shows, the *MATE* setting has the best results during pre-training on the AKCES dataset, and the *Typical Error* setting has very similar results. The *MorphoDiTa* and *Aspell* settings are less than ten percentage points worse.

Table 3.3 shows the evaluation on the AKCES and GECCC datasets after 45 epochs of pre-training. The results show that adding typical Czech errors significantly improves the model, especially the MATE setting. We also select the best checkpoints during every pre-training and compare them to each other on the AKCES dataset.

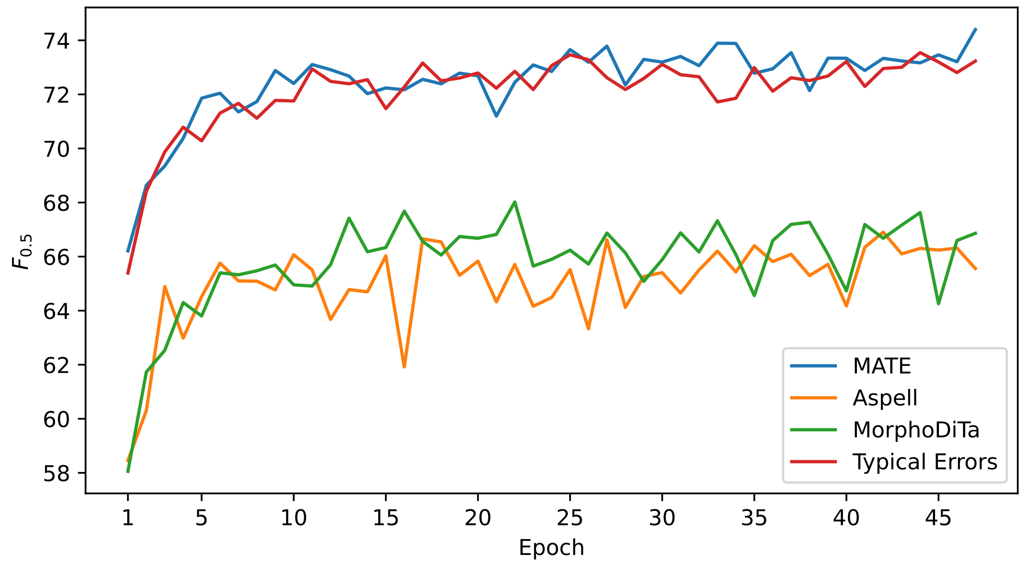

Figure 3.1: Comparison of methods for synthetic errors generation during pre-training, evaluated on the AKCES dataset using the $F_{0.5}$-score.

	45th Epoch		The Best	
Setting	AKCES	GECCC	Epoch	AKCES
Aspell	67.22	55.41	65	67.11
MorphoDiTa	64.19	53.24	22	67.77
Typical Errors	73.58	**65.55**	44	**74.52**
MATE	**74.23**	**65.55**	47	74.11

Table 3.3: Comparison $F_{0.5}$-score of settings.

3.2 Fine-tuning vs. Pre-training

The pre-training has a short growth period in the $F_{0.5}$-score until about the 10th epoch, after which it stagnates and does not change much. However, even if the performance on the AKCES development set does not increase further, the model might still improve its understanding of the language. We, therefore, perform experiments where we continue to pre-train the model further and test whether this affects the model, especially in the following fine-tuning. Only AKCES train data is used for fine-tuning, not mixed with synthetic data.

For each setting, we selected three checkpoints from the pre-training. We tried to choose them evenly, from the beginning, middle, and end of the pre-training. Each setting took a different amount of time due to the complexity of data generation or the currently available computing capacity. Therefore, each setting has differently selected checkpoints.

For *Aspell* setting, we start from the 45th, 60th, and 75th checkpoints. We can see that the long pre-training positively affects the following fine-tuning in this setting, even though the 60tth and 75th fine-tunings already have very similar courses (see Figure 3.2).

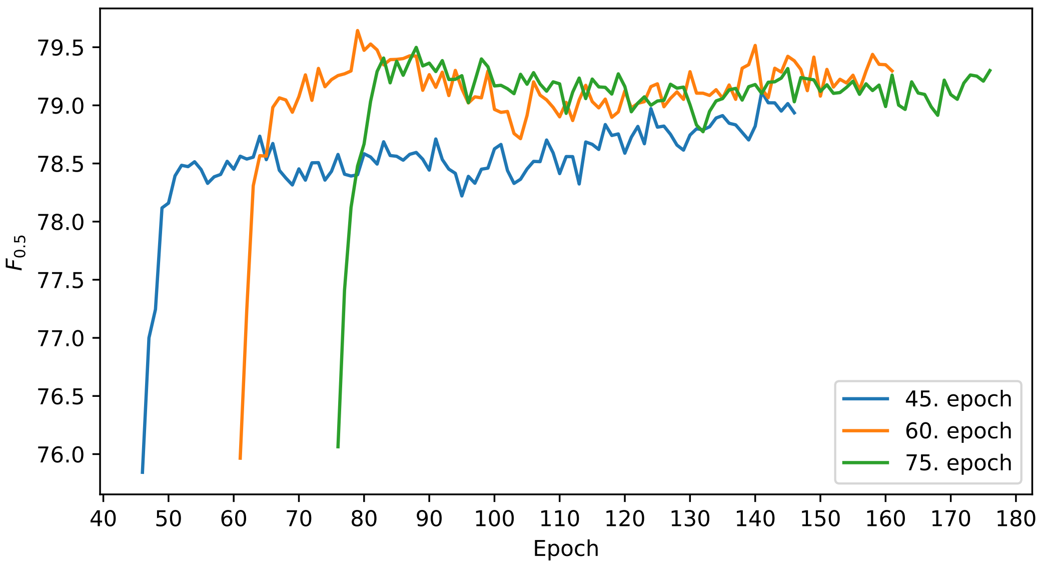

Figure 3.2: Fine-tunings from different checkpoints for setting *Aspell*. The $F_{0.5}$-score is evaluated on the AKCES dataset.

For *MorphoDiTa* settings, we select 22nd, 45th and 60th checkpoint. We can observe from Figure 3.3 that the time of pre-training helps, the difference between the 45th and 60th checkpoint is noticeable, but it is not such a difference as between 22nd and 45th checkpoint.

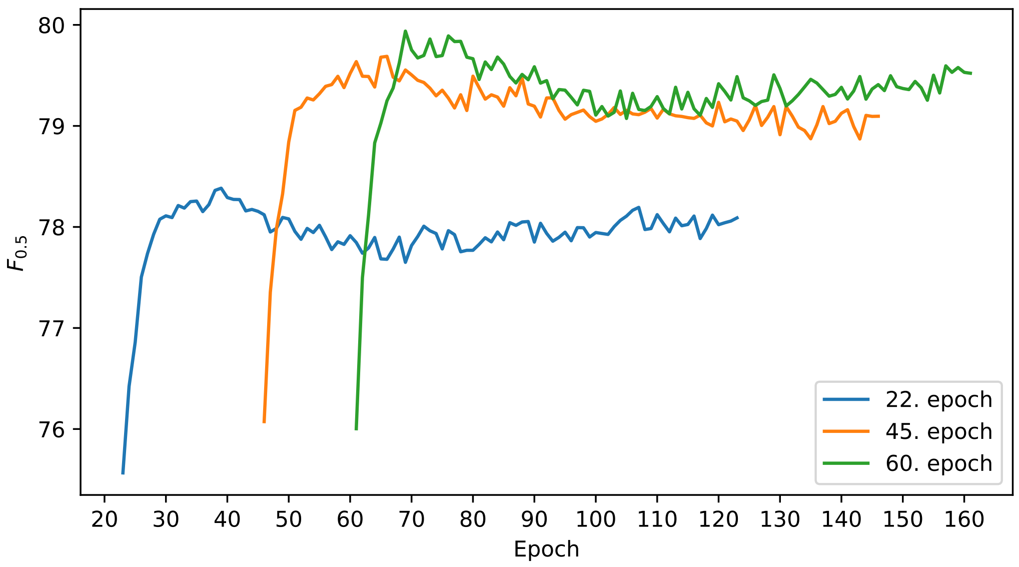

Figure 3.3: Fine-tunings from different checkpoints for setting *MorphoDiTa*. The $F_{0.5}$-score is evaluated on the AKCES dataset.

For *Typical Errors*, we pre-train up to the 45th epoch, so we start from the 9th, 30th, and 45th checkpoint. Figure 3.4 shows how the long pre-training improves fine-tuning performance here as well. We also see that the difference between the fine-tuning from the 9th epoch and the 30th is large, while the difference between the fine-tuning from the 30th and 45th is significantly smaller.

The *MATE* setting (Figure 3.5) has results similar to *Aspell*. Here, it is also confirmed that a long pre-training helps to improve the fine-tuning performance; however, we can see that the difference between the 45th and 70th epoch is already minimal (similar to the Figure 3.2 for Aspell between the 45th and 60th epoch).

The results show that long pre-training has a positive effect on fine-tuning performance. However, in our experiments, fine-tuning seems to start to stagnate from the 50th epoch, even with longer pre-training. In any case, even a longer pre-training, more than 50 epochs (less than 70 epochs), does not make the model noticeably worse.

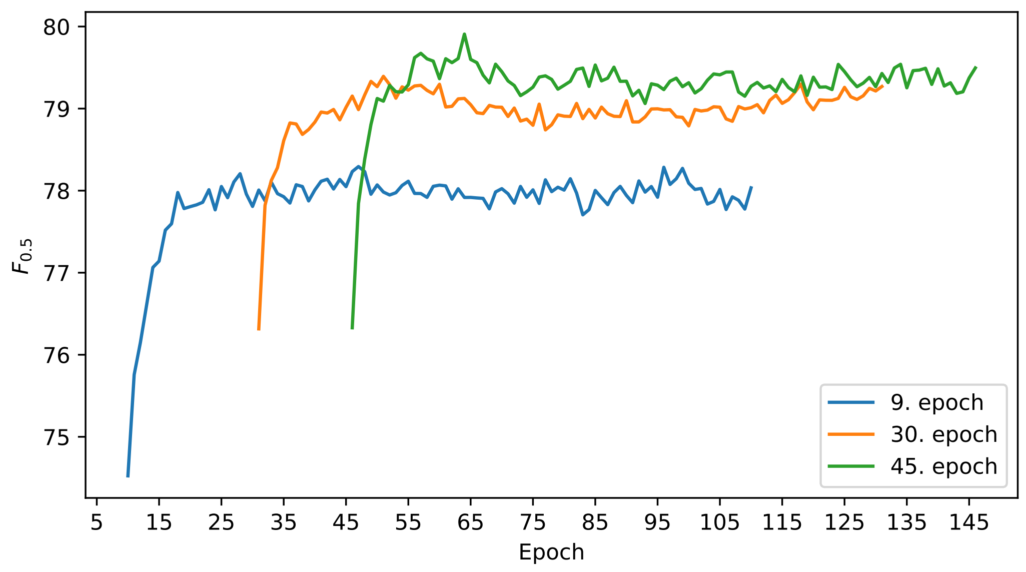

Figure 3.4: Fine-tunings from different checkpoints for setting *Typical Errors*. The $F_{0.5}$-score is evaluated on the AKCES dataset.

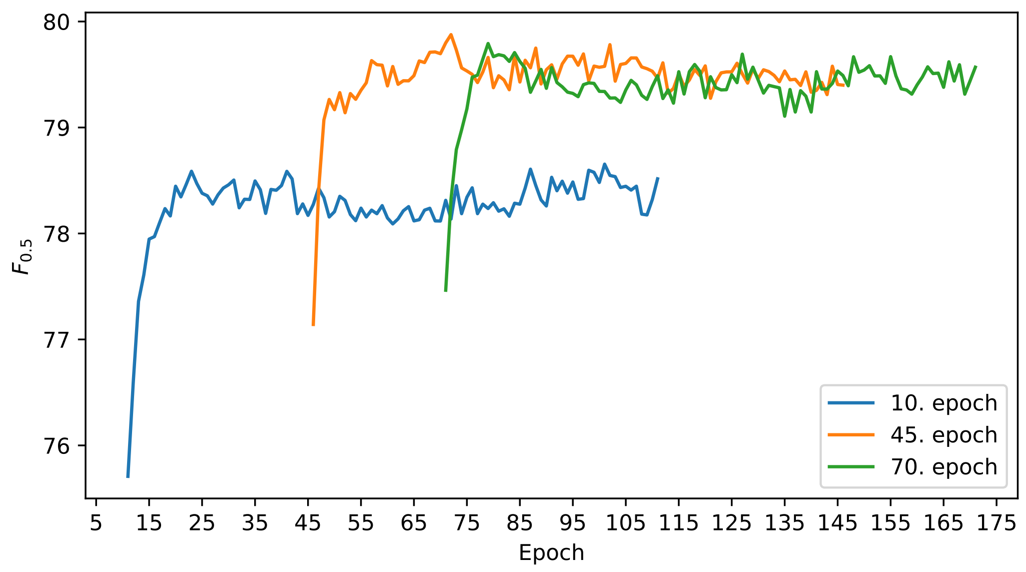

Figure 3.5: Fine-tunings from different checkpoints (epochs) for setting *MATE*. The $F_{0.5}$-score is evaluated on the AKCES dataset.

3.3 Fine-tuned Model – AKCES

We now search for the best fine-tuning method for the AKCES dataset. Although MATE is the best pre-training setting, is it the same in the following fine-tuning? This question is justified because the best results in the pre-training stage do not guarantee the best results in the fine-tuning stage. This shows us the fact that the length of pre-training has an effect on fine-tuning (previous Section 3.2). It could be the same with the fine-tuning method.

Another aspect we examine is the effect of data mixing on fine-tuning. We mix data only during fine-tuning by combining synthetic data and data from the annotated dataset (AKCES or GECCC). We run experiments with three ratios to mix them: 0:1 (no synthetic data), 2:1, and 5:1.

For each setting, we use its pre-trained model and always start fine-tuning from the 45th checkpoint. This also applies to all ratios, where synthetic data is generated in the same way as in pre-training.

The first is the ratio 0:1 (see Figure 3.6), where we see a rapid improvement in all settings. The *Aspell* setting loses out to the others, and the *MATE* setting looks best, outperforming both *Aspell* and *MorphoDiTa*.

The second is the 2:1 ratio (see Figure 3.7); improvement is slower. *Aspell* loses again, but not as much as in the previous ratio. The *MATE* setting is again the best and has the highest $F_{0.5}$-score in most epochs.

The last one is a ratio of 5:1 (see Figure 3.8). The improvement grows the slowliest, and again, the *MATE* setting is the best, the *Typical Errors* and *MorphoDiTa* settings are also good, and the last one is *Aspell*.

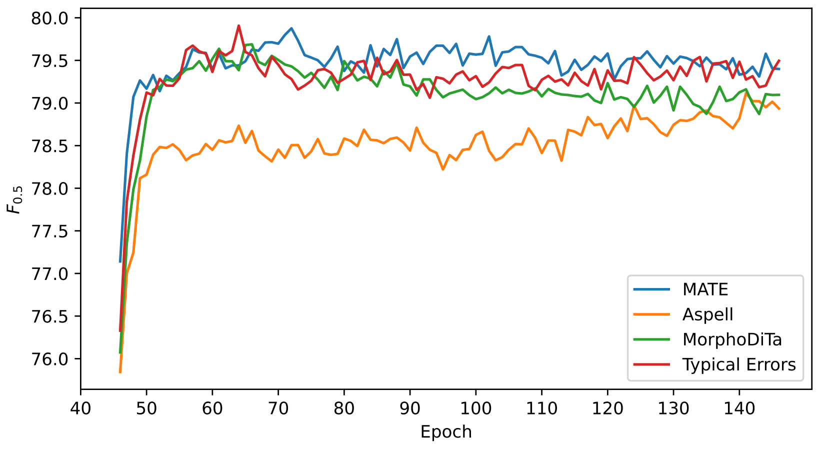

Figure 3.6: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 0:1. The $F_{0.5}$-score is evaluated on the AKCES dataset.

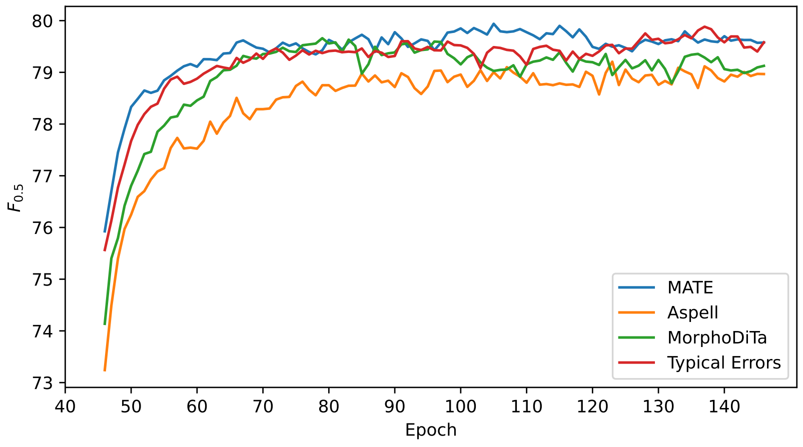

Figure 3.7: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 2:1. The $F_{0.5}$-score is evaluated on the AKCES dataset.

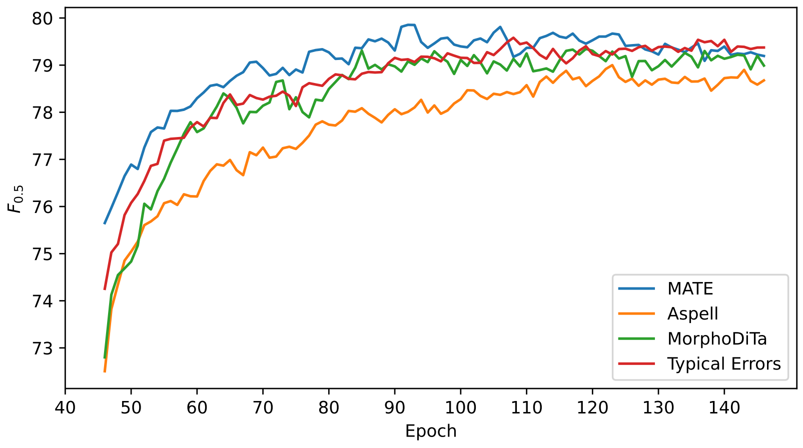

Figure 3.8: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 5:1. The $F_{0.5}$-score is evaluated on the AKCES dataset.

The *MATE* setting is the best for each mixing ratio and also produces the best results after pre-training. Thus, we select this setting as the best option to fine-tune the AKCES dataset.

Figure 3.9 compares all mixing ratios for the MATE setting. All of them achieve good results, so we find the best checkpoints on the AKCES dev and evaluate them on the AKCES test. Table 3.4 shows us that the ratio 0:1 performs the best. It is interesting because Náplava et al. [2022] claim that the combination of synthetic and annotated data performs better because the model quickly overfits them on clear annotated data. However, fine-tuning remains relatively stable for us.

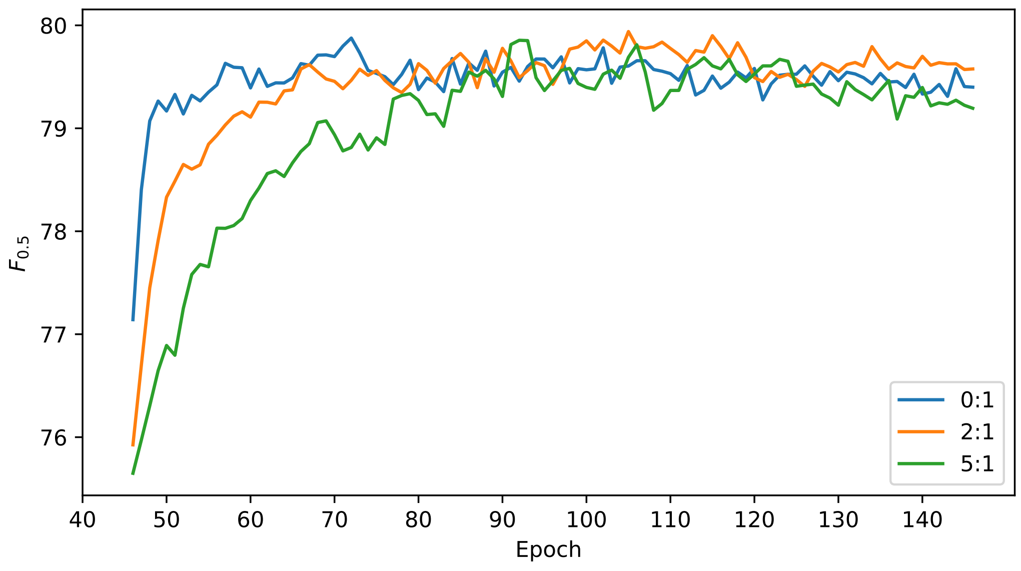

Figure 3.9: Comparison of fine-tunings of setting *MATE* with different ratios. The $F_{0.5}$-score is evaluated on the AKCES dataset.

Ratio	Epoch	$F_{0.5}$-score
0:1	72	**81.19**
2:1	105	80.98
5:1	92	80.24

Table 3.4: Comparison of fine-tunings of setting *MATE* with different ratios. The $F_{0.5}$-score is evaluated on the AKCES dataset.

3.4 Finetuned Model – GECCC

Just as we search for the best setting for the AKCES dataset, we search for it for the GECCC dataset. We start with the same approach of comparing all settings and ratios for fine-tuning. The settings remain the same, and again, we use mixing ratios of 0:1 (GECCC only), 2:1, and 5:1.

For each setting, we use its pre-trained model again and always start fine-tuning from the 45th checkpoint. This also applies to all ratios, where synthetic data is generated in the same way as in pre-training.

The best setting for a ratio of 0:1 is the *MATE* setting, which rises quickly (see Figure 3.10). The setting *Typical Error* has similar progress but does not achieve such performance. *MorpohoDiTa* is already worse than the two above, and below that is *Aspell*, which has the worst score in all epochs.

For the 2:1 ratio, the progress is already different (see Figure 3.11). In the beginning, *Typical Errors* looks best, but *MATE* beats it from the 70th epoch. *MorphoDiTa* and Aspell are about one percentage point worse, and from the 100th epoch, *MorphoDiTa* has a worse score than *Aspell*.

For the 5:1 ratio, the increase of curves is the mildest (see Figure 3.12). Again, the best settings are *MATE* and *Typical Errors*, which are comparable. Both *MorphoDiTa* and *Aspell* are again about one percentage point worse, but in recent epochs, *Aspell* outperforms *MorphoDiTa*.

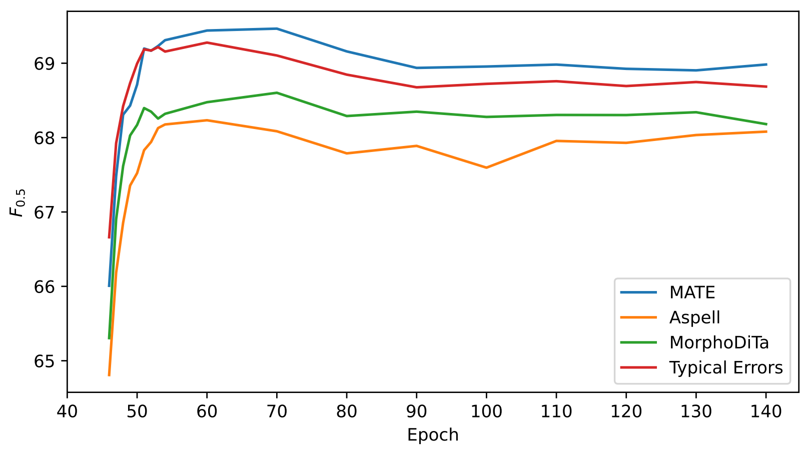

Figure 3.10: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 0:1. The $F_{0.5}$-score is evaluated on the GECCC dataset.

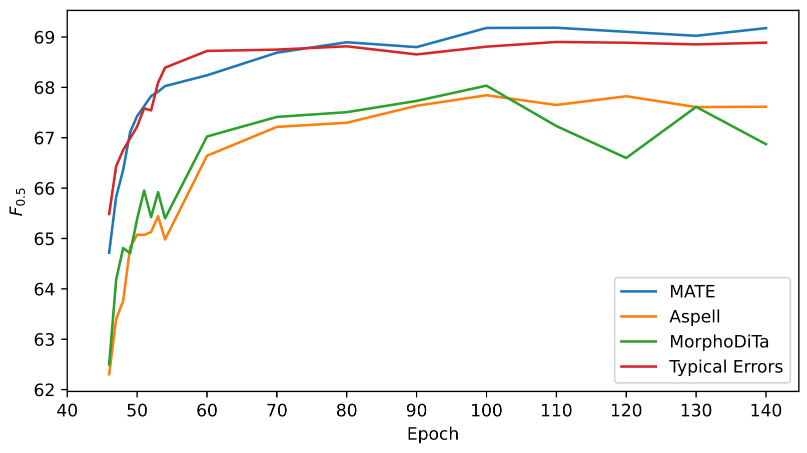

Figure 3.11: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 2:1. The $F_{0.5}$-score is evaluated on the GECCC dataset.

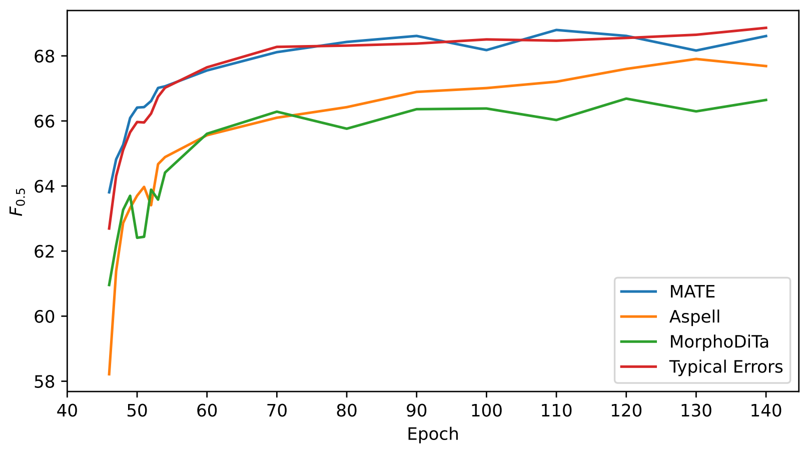

Figure 3.12: Comparison of fine-tunings with different methods to generate synthetic errors with ratio 5:1. The $F_{0.5}$-score is evaluated on the GECCC dataset.

The *MATE* setting is the best and outperforms the other three on the GECCC dataset. It, along with the Typical Errors setting, is the best even after pre-training. Adding typical Czech errors significantly improves the model's performance in pure pre-training and following fine-tuning.

In Figure 3.13 we compare the mix ratios for the *MATE* setup. Ratio 0:1 reaches the highest $F_{0.5}$-score and Table 3.5 shows that ratio 0:1 has the best performance.

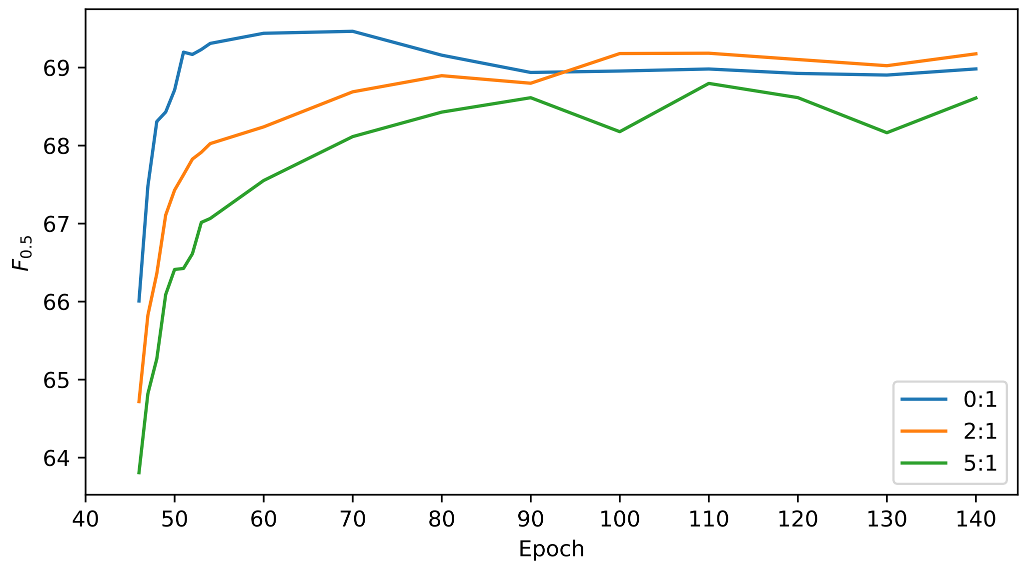

Figure 3.13: Comparison of fine-tunings of setting *MATE* with different ratios. The $F_{0.5}$-score is evaluated on the GECCC dataset.

Ratio	Epoch	$F_{0.5}$-score
0:1	70	**72.25**
2:1	100	72.08
5:1	110	71.86

Table 3.5: Comparison of fine-tunings of setting *MATE* with different ratios. The $F_{0.5}$-score is evaluated on the GECCC dataset.

3.4.1 Domains

The GECCC dataset is composed of four domains: Natives Formal (NF), Natives Web Informal (NWI), Romani (R), and Second Learners (SL). This raises the question of what fine-tuning looks like on individual domains and their evaluation.

We perform four fine-tunings; we fine-tune each domain separately. Figure 3.14 shows the progress of fine-tunings, and Table 3.6 compares the evaluation on GECCC. We use the *MATE* settings with its pre-trained model for each fine-tuning and always start from the 55th checkpoint.

Fine-tuning the Romani domain performs best, probably due to the more extensive training data size (27 824 sentences), which can cover a larger range of errors. The size of the representation of the Romani domain is smaller in the development and test data, but the higher error rate of these samples compensates for this.

Fine-tuning the domain Natives Web Informal has the second highest $F_{0.5}$-score on the development data. This domain already has a smaller training data size than the Romani domain, but still has a significant error rate and size of representation in the development and test data. However, when evaluated on the test data, the NWI domain is worse than the SL.

The third is fine-tuning Second Learners (on the development data), which is quite surprising given the size of the training data (30 812), the error rate, and the representation size in the development and test data. The lower performance than Natives Web Informal may be due to a lower error rate in the training data, or the training data does not have such a large overlap of errors with the development or test set as with Natives Web Informal. As mentioned above, when evaluated on the test dataset, it outperforms NWI.

The last one is fine-tuning the Natives Formal domain, which is unsurprising because Natives Formal has small training data and the lowest error rate in the development and test data.

Fine-tuning Data	Size	$F_{0.5}$-score
Natives Formal	4 060	67.71
Natives Web Informal	6 977	69.79
Romani	24 824	**71.44**
Second Learners	30 812	70.02
GECCC	66 673	72.25

Table 3.6: Comparison of fine-tunings using one specific domain data, evaluated on the GECCC dataset.

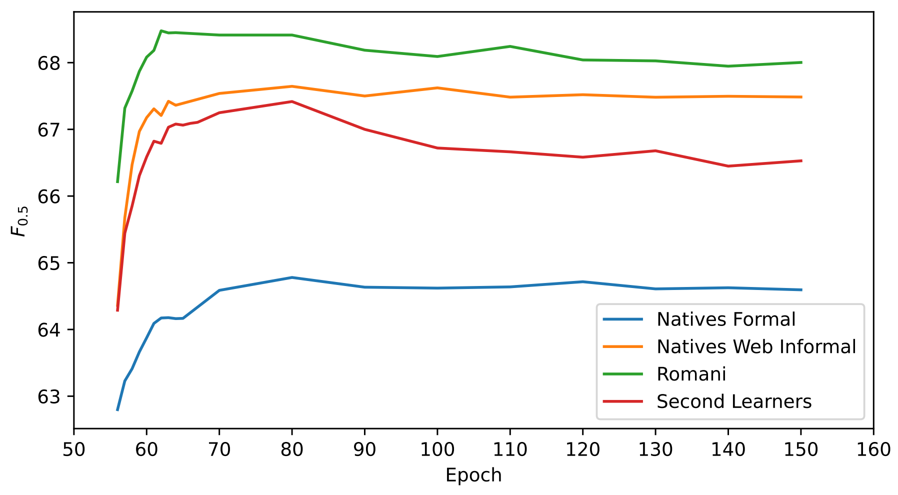

Figure 3.14: Comparison of fine-tunings of domains. The $F_{0.5}$-score is evaluated on the GECCC dataset.

We evaluate domain fine-tunings on the AKCES dataset (see Figure 3.15), where the best fine-tuning is Romani and then fine-tuning Second Learners, which is unsurprising since AKCES consists of Romani and Second Learners domains. We find the results in Romani to be the best, even though Second Learners have the best $F_{0.5}$-score in Table 3.7 because Romani outperforms Second Learners each epoch in the evaluation on development data. Natives Formal and Natives Web Informal fine-tunings are approximately four percentage points worse.

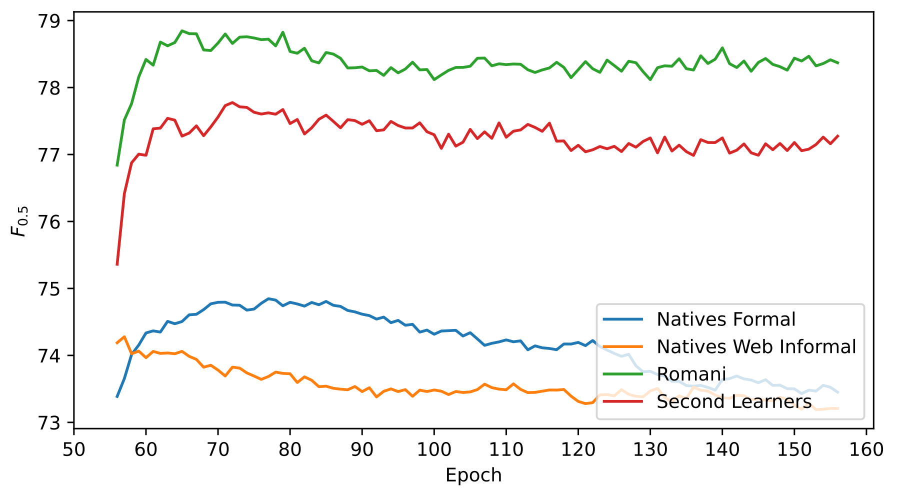

Figure 3.15: Comparison of domain fine-tunings evaluated on the AKCES dataset.

Fine-tuning Data	Size	$F_{0.5}$-score
Natives Formal	4 060	75.77
Natives Web Informal	6 977	74.96
Romani	24 824	79.35
Second Learners	30 812	**79.41**
AKCES	42 210	81.19

Table 3.7: Comparison of fine-tunings using one specific domain data, evaluated on the AKCES dataset.

Finally, we also evaluate each domain separately (see Figures 3.16, 3.17, 3.18 and 3.19), where it is confirmed that fine-tuning solely on the in-domain data leads to the best results on each domain, where it is confirmed that fine-tuning solely on the in-domain data leads to the best results on the domain. Here, it is also possible to see the mutual relations of the domains. For example, fine-tuning Romani has the best results for evaluating the Romani domain (see Figure 3.18), but it also has good results for evaluating the Natives Formal domain (see Figure 3.16). The Romani domain probably covers the most errors from Natives Formal compared to Natives Web Informal or Second Learners.

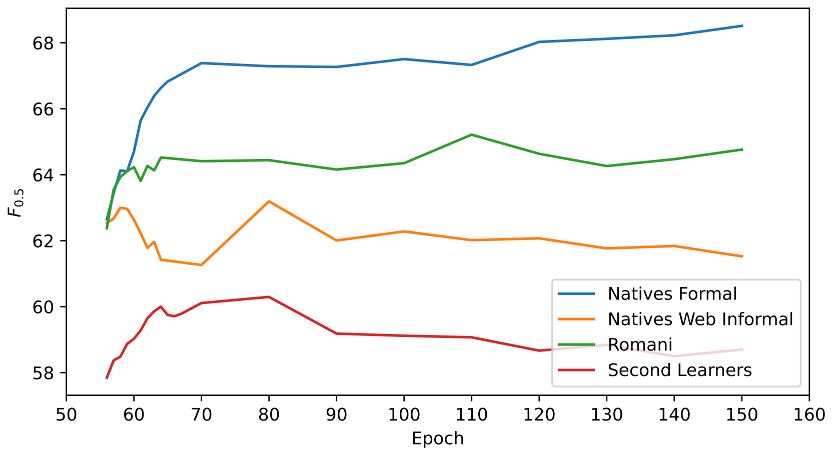

Figure 3.16: Comparison of domain fine-tunings evaluated on the Natives Formal domain.

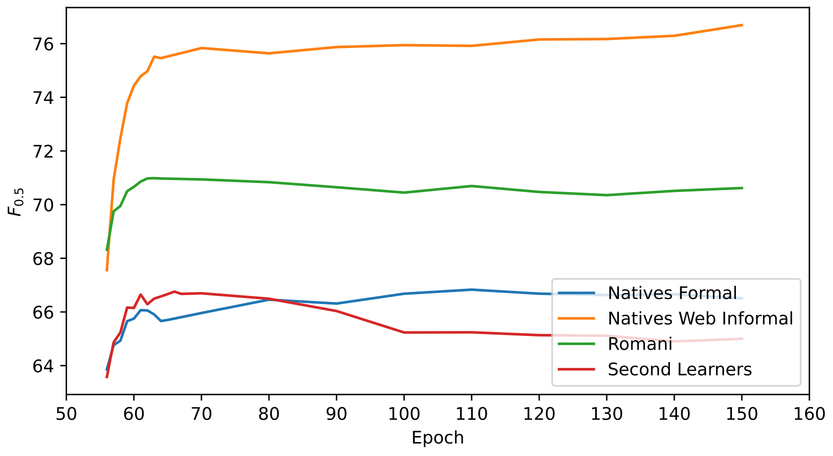

Figure 3.17: Comparison of domain fine-tunings evaluated on the Natives Web Informal domain.

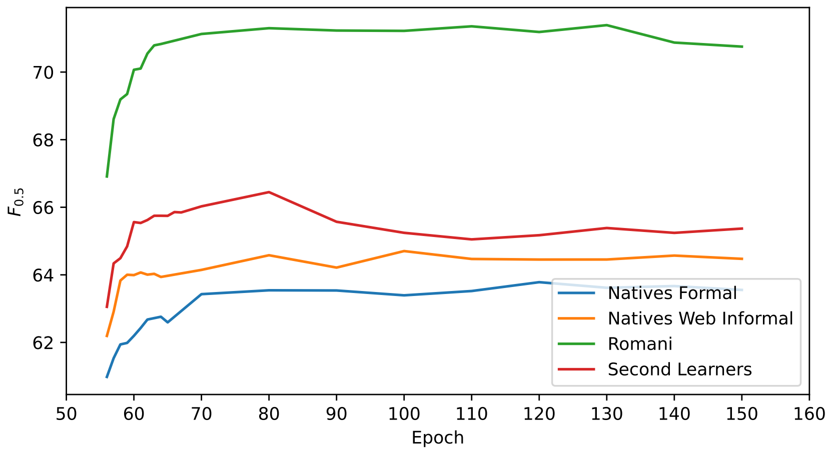

Figure 3.18: Comparison of domain fine-tunings evaluated on the Romani domain.

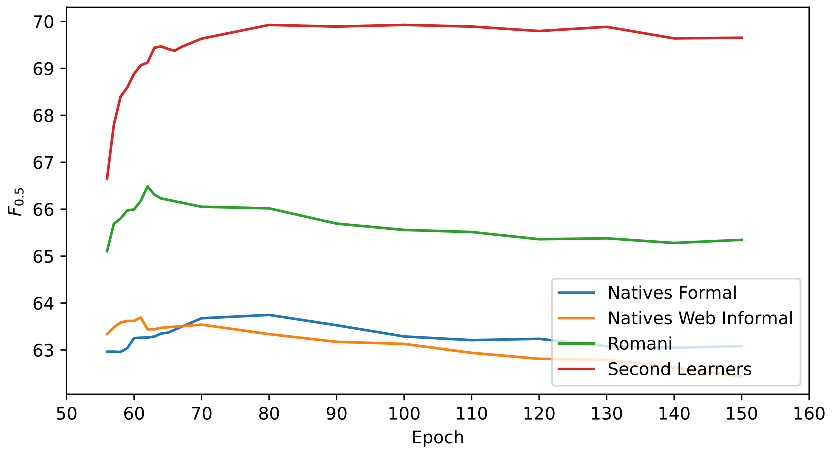

Figure 3.19: Comparison of domain fine-tunings evaluated on the Second Learners domain.

Table 3.8 shows the improvement of models fine-tuned on individual domains over the model that is fine-tuned on the entire GECCC dataset. From the results, we deduce that fine-tuning on the specific domain does not worsen the performance of the model on the specific domain, but rather improves it. For all domain fine-tunings, the results for their specific domain are better than when fine-tuned with the entire GECCC training set, except for the Natives Web Informal domain, where the difference compared to GECCC is minimal. These observations could be beneficial in the building of narrowly focused systems.

Evaluation Data	Fine-tuning Data	$F_{0.5}$-score	Relative Improvement
Natives Formal	GECCC	70.46	
Natives Formal	Natives Formal	71.87	+1.41
Natives Web Informal	GECCC	75.91	
Natives Web Informal	Natives Web Informal	75.83	-0.08
Romani	GECCC	73.04	
Romani	Romani	74.75	+1.71
Second Learners	GECCC	71.82	
Second Learners	Second Learners	72.64	+0.82

Table 3.8: Comparison of fine-tunings using one specific domain data with the model fine-tuned on the whole GECCC dataset. The relative improvement is evaluated against the performance of the fine-tuned model on GECCC.

3.4.2 Oversampling

By evaluating individual domains, we find that each domain contributes to the final score to a certain extent. The distribution of domain contribution is not uniform; each domain contributes differently. Therefore, we examine how the oversampling of individual domains affects performance.

We go through each domain, calculate its size, and raise it to the power of factor ($size^{factor}$). The resulting value is the weight with which we select a random sample from the corresponding domain. It is shown in detail in Figure 3.20.

```
n = number of samples
total_size = size of all domains
weights = {}

for domain in domains:
    size = size of domain
    weight = pow(size, factor)
    weights[domain] = weights

probabilities = {}
for domain in domains:
    probability = weight / sum(weights)
    probabilities[domain] = probability

sentences = []
for i in 1..n:
    domain = select_domain(probabilities)
    sentence = randomly_choose_sentence(domain)
    sentences.append(sentence)

return sentences
```

Figure 3.20: Language domain oversampling.

In this way, we create 5 datasets using the factors 0.0 (uniform domain distribution), 0.25, 0.5, 0.75, and 1.0 (domain distribution of the original GECCC).

We use the pre-trained model with setting *MATE* for each fine-tuning and always start from the 55th checkpoint.

Figure 3.21 shows that oversampling fine-tuning with factors of 0.25 and 0.75 achieve the best results. Factors 0.5 and 1.0 are already a little worse, and factor

0.0 is the worst. In Table 3.9 we can see a comparison of the best checkpoints, where the factor 0.25 reaches the best $F_{0.5}$-score.

Table 3.9 also shows the evaluation on individual GECCC domains. The 0.25 factor performs the best on Natives Formal and Natives Web Informal domains. We hypothesize these results are caused by the fact that the 0.25 factor oversamples the small domains more than the large ones. On the contrary, on the Second Learners and Romani domains, the best factor is 0.75, which oversamples more the larger domains.

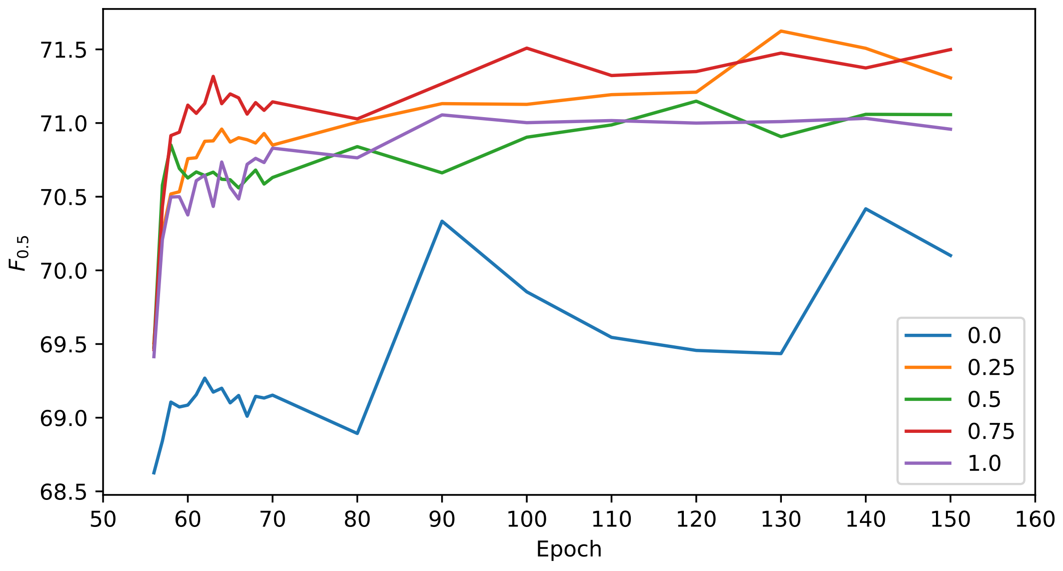

Figure 3.21: Comparison of oversampling factors used during fine-tunings on GECCC. The $F_{0.5}$-score is evaluated on the GECCC dataset.

Factor	Epoch	$F_{0.5}$-score	NF	NWI	SL	R
0.00	140	71.82	72.29	75.78	68.96	71.67
0.25	130	**73.52**	**73.36**	**77.19**	71.13	73.36
0.50	120	72.96	71.93	76.09	71.11	72.80
0.75	100	73.32	71.19	75.93	**71.78**	**73.55**
1.00	90	73.09	70.40	75.91	71.52	73.27

Table 3.9: Results of oversampling factors used during fine-tunings on GECCC. The $F_{0.5}$-score is evaluated on the GECCC dataset, and NF, NWI, SL, and R are evaluations on language domains.

3.5 Corpora Comparison

Another important part of our GEC system is the corpus from which noisy sentences are generated. Our assumption is that the corpus contains no errors and only our pipeline introduces them. Unfortunately, we can never be sure that the corpus is completely error-free, especially for large texts. Thus, we examine how the purity of the data will affect our model, both in the pre-training and fine-tuning stage. We train on four corpora mentioned in Section 1.4.2.

3.5.1 Pre-training

We pre-train four models, one for every source of clean data, from which noisy data are generated during training. Otherwise, all other settings will be identical (*MATE* settings, base size Transformer model).

Figure 3.22 shows that we pre-train each model for at least 50 epochs. The SYN corpus has the highest $F_{0.5}$-score on the AKCES dataset and on the GECCC dataset (see Table 3.10), probably because it is a synchronous Czech national corpus emphasizing text correctness. Its most significant difference from the others is in the Natives Formal domain, where it outperforms the others by up to 8 percentage points. Corpus News 2019 is the second best, even with a better F-score on the Natives Web Informal domain than SYN-v4. The third is the Wikipedia corpus, which lost about five percentage points to the best SYN-v4. The last is the Common Crawl corpus, which is probably more noisy; its most significant drop is in the Natives Web Informal domain.

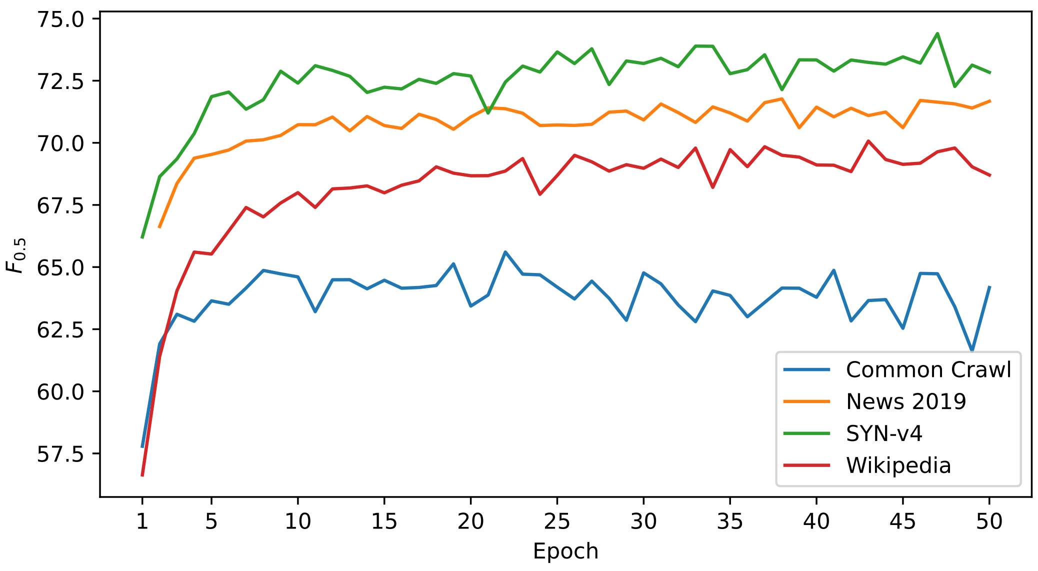

Figure 3.22: Pre-trainings of different corpora evaluated on the AKCES dataset.

Corpus	AKCES	GECCC	NF	NWI	R	SL
Common Crawl	67.44	55.27	56.94	44.59	53.01	60.30
News 2019	72.36	63.54	58.19	**62.94**	63.02	65.16
SYN-v4	**74.11**	**64.00**	**66.16**	59.94	**63.62**	**65.91**
Wikipedia	69.89	60.23	55.28	55.43	61.75	63.19

Table 3.10: Corpora comparison in the pre-training stage evaluated on the AKCES dataset in the best epoch and on the GECCC dataset in the 50th epoch (All domains are also evaluated in the 50th epoch).

3.5.2 Fine-tuning

We also run fine-tuning for each corpus from its 50th checkpoint on the AKCES and GECCC datasets. For AKCES fine-tuning, we use only AKCES data (ratio 0:1), and for GECCC fine-tuning, we use oversampled GECCC data with a factor of 0.25.

Figures 3.23 and 3.24 show that fine-tuning based on pre-training with the SYN-v4 corpus achieves the best results, both on the AKCES dataset and the GECCC dataset. The second best is fine-tuning based on pre-training with the News 2019 corpus, which has good results. The third best is a bit surprisingly Common Crawl, which outperforms Wikipedia on both AKCES and GECCC. The Common Crawl was worse than Wikipedia during the entire pre-training but since Common Crawl is the largest (more than 7 times larger than Wikipedia, which is the smallest), the pre-trained model was able to learn more about the language itself (even if not about the grammar error correction, given that the Common Crawl data is very noisy), which was then beneficial during fine-tuning.

In Table 3.11, we compare all fine-tunings in their best checkpoints; here again, Common Crawl beats Wikipedia, but only by a small margin.

Corpus	Epoch	AKCES	Eepoch	GECCC
Common Crawl	133	79.73	90	71.93
News 2019	63	79.76	130	72.52
SYN-v4	65	**81.32**	130	**72.97**
Wikipedia	99	79.25	150	70.94

Table 3.11: Corpora comparison in fine-tuning. The $F_{0.5}$-score is evaluated on the AKCES dataset and on the GECCC dataset.

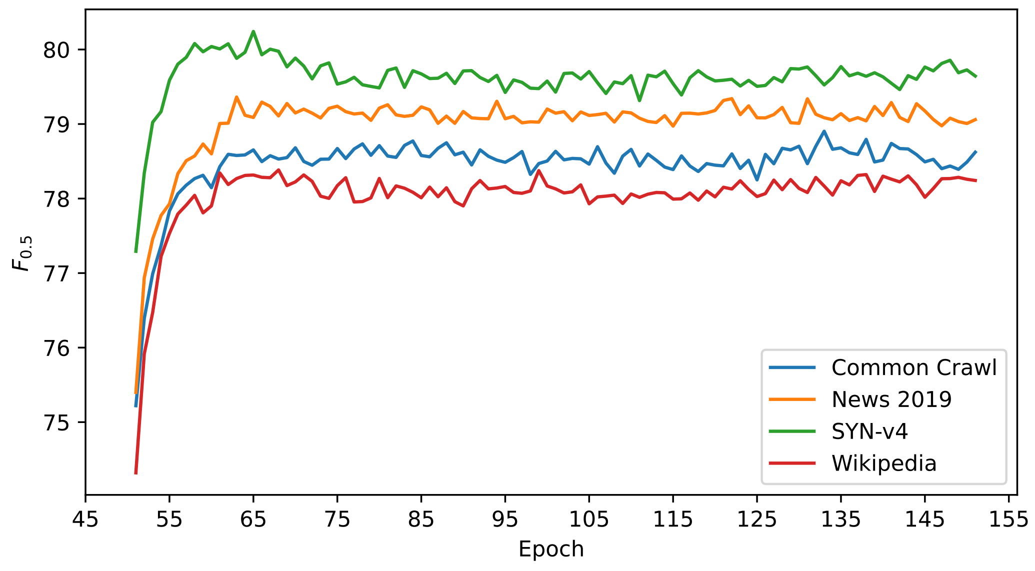

Figure 3.23: Fine-tuning of different corpora on AKCES. The $F_{0.5}$-score is evaluated on the AKCES dataset.

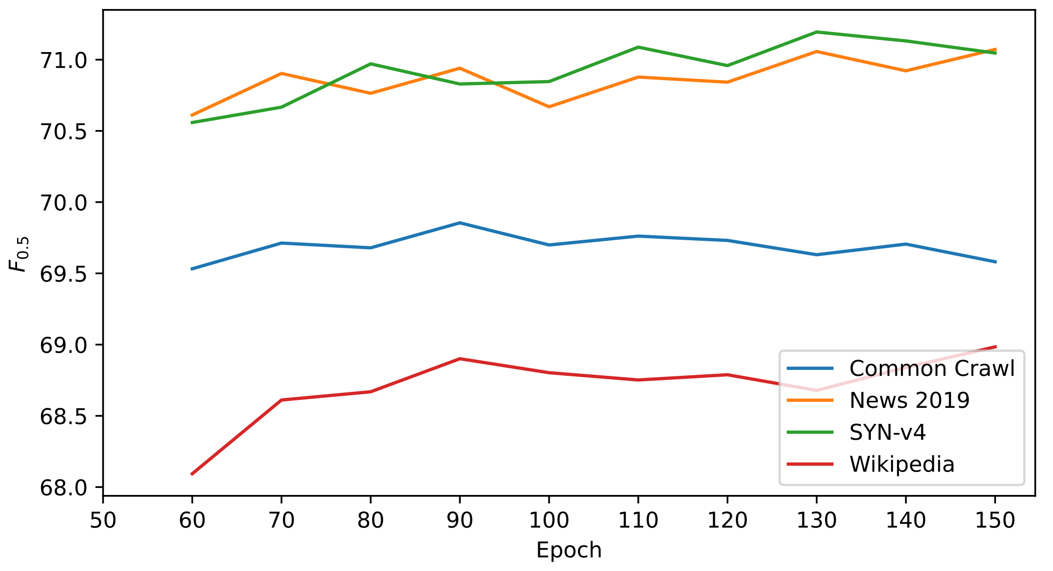

Figure 3.24: Fine-tuning of different corpora on GECCC. The $F_{0.5}$-score is evaluated on the GECCC dataset.

3.6 Model Comparsion

In this section, we examine how enlarging models improves performance. We run experiments with three different sizes of our Transformer model: small, base, and large (see Table 3.12). For our large model, we use half the batch size and train on two GPUs, and for the small model, we leave the settings the same as for the base size. In addition to enlarging the model, we also try to use the already pre-trained MT5 model base size [Xue et al., 2021].

Model	Size	Dim	Layers	Heads	FFN	Params
Transformer	small	256	2	4	1 024	10×10^6
	base	512	6	8	2 048	65×10^6
	large	1 024	6	16	4 096	213×10^6
MT5	base	768	12	12	2 048	580×10^6

Table 3.12: Hyper-parameters and size of the Transformer model and the MT5 model.

MT5 Configuration

We use two GPUs and the Adafactor optimizer [Shazeer and Stern, 2018] with a learning rate of 5e-05. The batch size is 24, the epoch length is 341 333 steps ($2 \cdot 24 \cdot 341\,333 = 16\,384\,000$ samples per epoch), and the maximal length of a sample is 128 tokens. We use the same settings for both the pre-training and fine-tuning stages.

3.6.1 Pre-training

The length of the pre-training stage is different for each model. Figure 3.25 shows that Transformer-large performs best in pre-training, but MT5-base reaches a very similar performance. The Transformer-base model is a little worse, and the Transformer-small model performs the worst. The results are relatively consistent with intuition, namely that large models perform better. However, the difference between the large and base variants is not great. The pre-training stage is time-consuming; for the Transformer-small model, the epoch takes 8 hours, for the base size 12 hours, and for the large even 28 hours. However, the MT5-base model takes the longest, with one epoch lasting 96 hours.

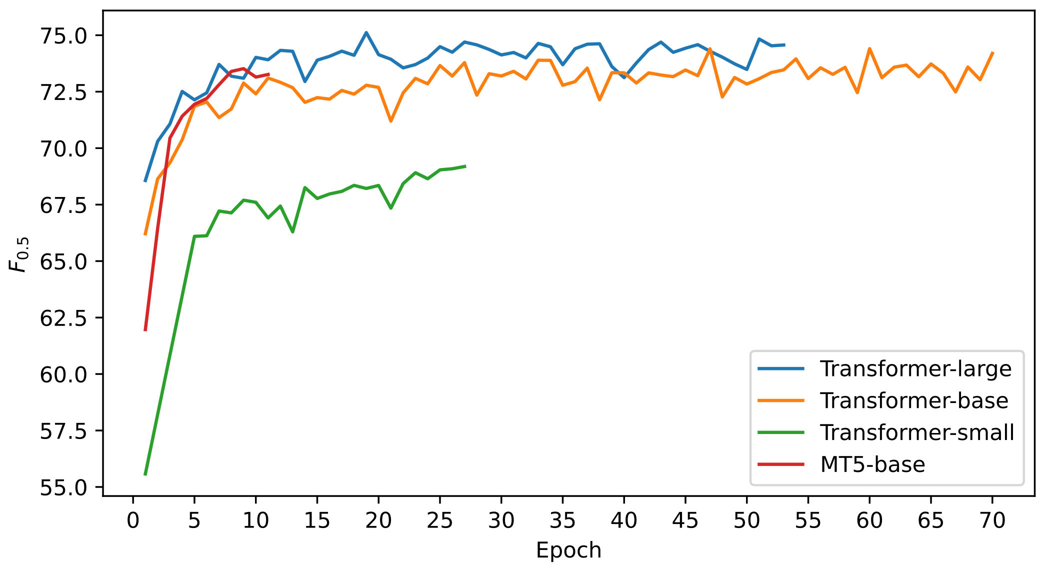

Figure 3.25: Comparison of pre-trainings with different models, evaluated on the AKCES dataset.

3.6.2 Fine-tuning

Again, we divide the fine-tuning stage for the AKCES dataset and the GECCC dataset. For AKCES fine-tuning, we use only AKCES data (ratio 0:1), and for GECCC fine-tuning, we use oversampled GECCC data with a factor of 0.25.

Figures 3.26 and 3.27 show that the Transformer-large and MT5-base models achieve the best results. They achieve a comparable $F_{0.5}$-score on both the AKCES dataset and the GECCC dataset. The Transformer-base model achieves slightly worse results, and the Transformer-small model is the worst. We can observe that size matters if we want to achieve the best results.

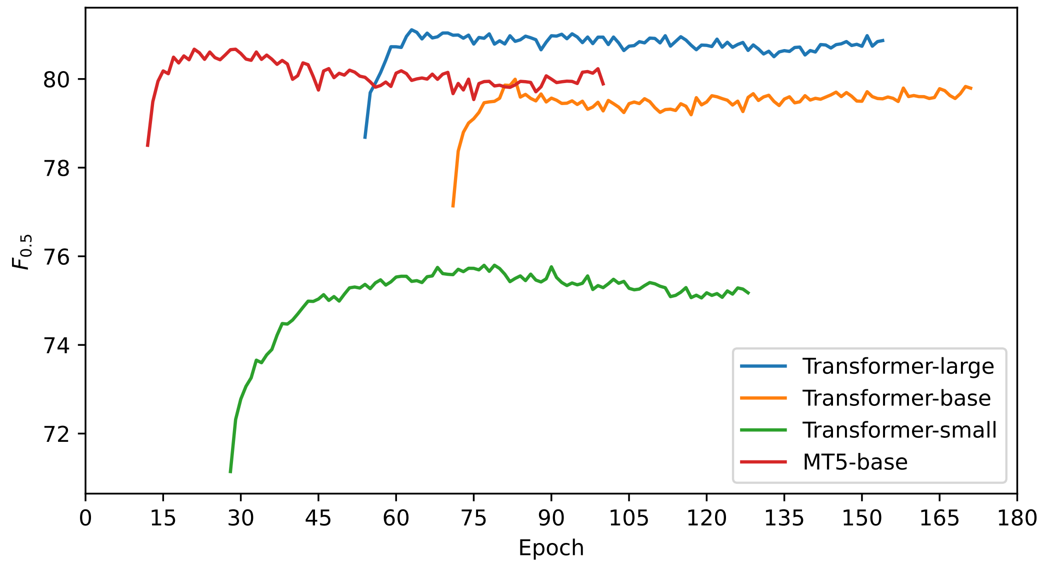

Figure 3.26: Comparison of different models fine-tuned on AKCES and evaluated on the AKCES dataset.

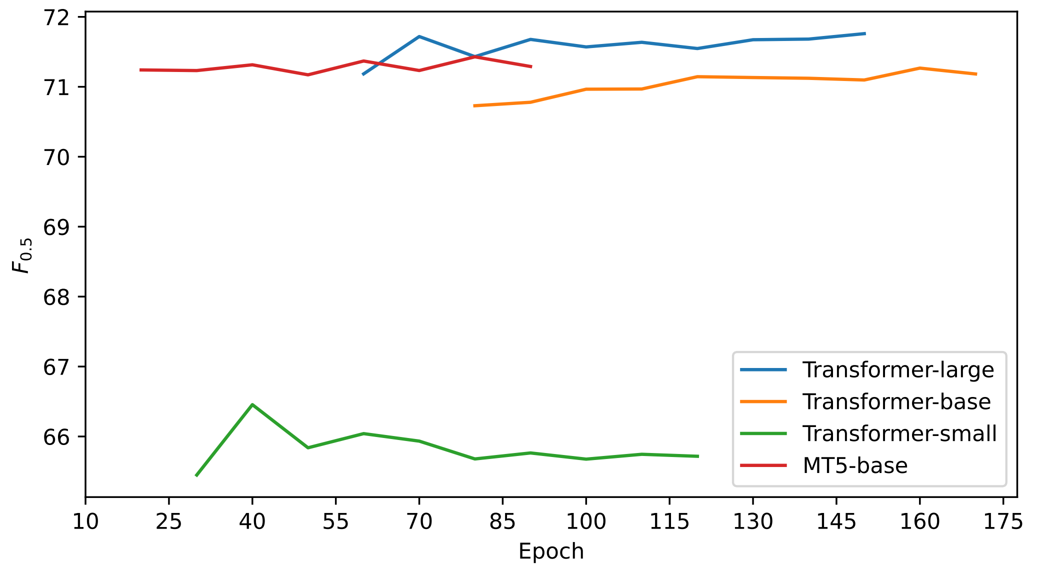

Figure 3.27: Comparison of different models fine-tuned on GECCC and evaluated on the GECCC dataset.

Final Evaluation

In Table 3.13, we present the results of our models and compare them with the results of other authors that achieve the best results for the Czech language.

We introduce models from Katsumata and Komachi [2020] and Náplava and Straka [2019], as well as a base size model from Rothe et al. [2021]. We outperform these three models with our Transformer-base model fine-tuned on GECCC, which reaches a higher $F_{0.5}$-score on both AKCES and GECCC datasets.

It is confirmed that the larger model achieves better scores. It can be seen that Rothe et al. [2021] has the best results at AKCES, but unlike us, they use a 13B model, while we have a 22 times smaller model and achieve the $F_{0.5}$-score that is only one point worse.

The model from Náplava et al. [2022] fine-tuned on the GECCC dataset achieves an $F_{0.5}$-score of 72.96. Our Transformer-large model fine-tuned on GECCC outperforms this model with an $F_{0.5}$-score of 73.92.

Model	Params	AKCES	GECCC
Rothe et al. [2021] base	*580M*	*71.88*	-
Rothe et al. [2021] xxl	*13B*	***83.15***	-
Katsumata and Komachi [2020]	*610M*	*73.52*	-
Náplava and Straka [2019] – AG finetuned	*210M*	*80.17*	*68.08*
Náplava et al. [2022] – GECCC finetuned	*210M*	-	*72.96*
Transformer-small – AKCES-finetuned	10M	76.41	66.19
Transformer-small – GECCC-finetuned	10M	75.19	68.03
Transformer-base – AKCES-finetuned	65M	81.25	71.03
Transformer-base – GECCC-finetuned	65M	79.95	73.32
Transformer-large – AKCES-finetuned	210M	81.56	72.12
Transformer-large – GECCC-finetuned	210M	80.66	**73.92**
MT5-base – AKCES-finetuned	580M	82.18	71.55
MT5-base – GECCC-finetuned	580M	79.98	73.50

Table 3.13: Comparison of different models evaluated on the AKCES and the GECCC datasets.

We analyze individual errors in Czech, as we defined them in Section 2.3. For each error, we count the number of occurrences and evaluate its Precision, Recall, and $F_{0.5}$-score using ERRANT [Bryant et al., 2017]. We evaluate individual errors on the GECCC dataset because it has a larger number of errors than the AKCES dataset.

Table 3.14 shows that errors related to diacritics are the most common in the data, and we also correct them very well. On the other hand, it is possible to improve our system on capitalization errors, which are not so well corrected.

In some cases, we are able to correct all occurrences of errors, especially for less frequent types of errors.

Error	Count	Precision	Recall	$F_{0.5}$
removing a comma	1753	72.6	86.3	75.0
removing diacritics	1607	81.3	93.5	83.5
adding diacritics	1604	83.6	91.4	85.0
sentence – first to uppercase	457	85.5	88.8	86.1
any word – first to lowercase	345	57.3	65.8	58.8
letter b/f/l/m/p/s/v/z followed by -i/-y	257	82.0	88.7	83.3
any word – first to uppercase	118	48.1	84.8	52.6
word mě/mně	39	83.0	100.0	85.9
prefix s-/z- or se-/ze	26	81.2	100.0	84.4
suffix -i/-y	21	69.2	85.7	72.0
preposition s/z	20	65.2	75.0	67.0
infix -mě-/-mně-	20	82.6	95.0	84.8
conditionals	12	100.0	25.0	62.5
sentence – first to lowercase	5	0.0	0.0	0.0
suffix -mě/-mně	5	100.0	80.0	95.2
word mi/my	4	40.0	100.0	45.5
suffix -bě/-bje	3	100.0	100.0	100.0

Table 3.14: Comparison of typical Czech errors evaluated on the GECCC dataset.

In Table 3.15, we introduce examples of sentence correction. We present examples from the Natives Formal and Second Learners domains that our model corrects.

Source Sentence	Corrected Sentence	Domain
Muj oblibeny kavárny je Luvur cáfe , ktery je blizko stanice Národni třida . (nebo pěšky od staníce Můstek)	**Moje oblíbená kavárna** je **Louvre Café** , kter**ý** je bl**í**zko **stanice Národní třída** (nebo pěšky od **stanice** Můstek) **.**	SL
Je uplně unikatní a věselé misto .	Je **to ú**plně unik**á**tní a v**e**selé m**í**sto .	SL
Když jsem čekala tam , nikdo přijel a vzal moji kabelku .	Když jsem **tam čekala** , **někdo** přijel a vzal moji kabelku .	SL
Víte co se stalo jednou Švejkovi ?	Víte **,** co se stalo jednou Švejkovi ?	NF
Pomocí farem a měst zvyšujte populaci.Optim ální je na jedno město , třicet farem .	Pomocí farem a měst zvyšujte **populaci . Optimální** je na jedno město třicet farem .	NF
Závěrečné slohová práce	**Závěrečná** slohová práce	NF

Table 3.15: Examples of source sentences corrected by our model. Domain describes from which domain example comes.

Conclusion

We presented the Grammatical Error Correction system for correcting the Czech language using the Neural Machine Translation approach. Our model is based on Transformer architecture. This approach requires an extensive amount of annotated data, which is insufficient for many languages, especially for Czech. Therefore, noisy sentences are generated from clean corpora to obtain the required amount of data. Náplava et al. [2022] have utilized the approach of generating synthetic noisy data from clean texts using simple language-independent rules. We have extended this approach by adding typical Czech errors to the automatic generation of noisy data.

We analyzed typical Czech errors and implemented a model training pipeline that includes the real-time generation of noisy data from clean corpora. This avoids creating the corpus once and allows us to change and adjust the data generation for the model easily.

We found that the length of the pre-training stage is important. Even if the GEC metrics stagnate in the pre-training stage, the subsequent fine-tuning performs better from the later checkpoints. This also holds for models that did not use typical Czech errors during learning.

We experimented with different data generation settings during both pre-training and fine-tuning stage:

- Settings that added typical Czech errors in the pre-training stage outperformed the settings without them, even in the following fine-tuning. We obtained the best pre-training setting that combines the adding typical Czech errors and simple language-independent rules with Aspell and MorphoDiTa tools.

- During fine-tuning, we experimented with mixing synthetic data and manually annotated data, which did not perform as well as fine-tuning on only annotated data. For fine-tuning on the GECCC dataset language containing four domains, we examined the oversampling of individual domains. The sizes of domains in GECCC vary, each contributing to the final score to a different extent. Changing the domain ratio can improve model performance. Oversampling with a factor of 0.25 is the best performer. This factor samples more from smaller domains than the original data distribution.

We compared different-quality corpora for the Czech language to see if this would affect the model's performance. We used four corpora to generate synthetic noise data. Models trained with higher-quality corpora performed significantly better in both the pre-training and fine-tuning stages.

We validated that larger models perform better than smaller. We increased the size of the Transformer model from base to large and also provided an evaluated small sized model that is significantly worse. Still, it could be the solution for narrowly focused tasks because the results are not so bad. The Transformer model of large size, trained from scratch, performed best, which even surpassed the already pre-trained MT5 model of base size (MT5 has an order of magnitude larger models). The pre-trained MT5 model was further pre-trained and fine-tuned by us in the same way as the Transformer model.

We presented models that achieve state-of-the-art results for the Czech language for comparable model size. The model MT5-base fine-tuned on AKCES achieves $F_{0.5}$-score of 82.18 on the dataset AKCES and the model Transformer-large fine-tuned on GECCC achieves $F_{0.5}$-score of 73.92 on the dataset GECCC.

In the Introduction, we stated the research questions that we answer here:

1. *"How can the GEC system in Czech be improved by synthetically adding typical Czech errors?"*

   Answer: *"The addition of typical Czech errors to the automatic generation of noisy data increases the model performance. In the pre-training and fine-tuning stages, models with typical Czech errors outperform the same models without them. In pre-training, the improvement in $F_{0.5}$-score can be up to eight percent points, and in fine-tuning, the improvement is around two percent points."*

2. *"How does the performance of the GEC system depend on clean data?"*

   Answer: *"To achieve the maximum performance of the model, it is necessary to have a clean corpus containing a minimum number of errors, preferably no errors. The corpus size is also important, but 15 million correct sentences overcome three times the number of noisier sentences."*

3. *"How does the model performance depend on its size?"*

   Answer: *"Larger models achieve better performance. The difference in the pre-training stage between small and base sizes is about 5 percentage points on the $F_{0.5}$-score, and between the base and large sizes, it is about half a percentage point. In the fine-tuning stage, the difference is about 4 percentage points between small and base, and 1 point between base and large."*

The implementation is released on GitHub at `https://github.com/petrpechman/czech_gec/tree/MasterThesis_PechmanPetr_2024`.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. Findings of the 2019 conference on machine translation (WMT19). In Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, André Martins, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Matt Post, Marco Turchi, and Karin Verspoor, editors, *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5301. URL `https://aclanthology.org/W19-5301`.

David C Blair. Information retrieval, 2nd ed. C.J. van rijsbergen. london: Butterworths; 1979: 208 pp. price: $32.50. *J. Am. Soc. Inf. Sci.*, 30(6):374–375, nov 1979.

Adriane Boyd, Jirka Hana, Lionel Nicolas, Detmar Meurers, Katrin Wisniewski, Andrea Abel, Karin Schöne, Barbora Štindlová, and Chiara Vettori. The MERLIN corpus: Learner language and the CEFR. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1281–1288, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2014/pdf/606_Paper.pdf`.

Chris Brockett, William B. Dolan, and Michael Gamon. Correcting ESL errors using phrasal SMT techniques. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 249–256, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220207. URL `https://aclanthology.org/P06-1032`.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993. URL https://aclanthology.org/J93-2003.

Christopher Bryant, Mariano Felice, and Ted Briscoe. Automatic annotation and evaluation of error types for grammatical error correction. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1074. URL https://aclanthology.org/P17-1074.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, London, England, 2 edition, September 2001.

Daniel Dahlmeier and Hwee Tou Ng. Better evaluation for grammatical error correction. In Eric Fosler-Lussier, Ellen Riloff, and Srinivas Bangalore, editors, *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572, Montréal, Canada, June 2012. Association for Computational Linguistics. URL https://aclanthology.org/N12-1067.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. Building a large annotated corpus of learner English: The NUS corpus of learner English. In Joel Tetreault, Jill Burstein, and Claudia Leacock, editors, *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 22–31, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://aclanthology.org/W13-1703.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Mariano Felice, Christopher Bryant, and Ted Briscoe. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In Yuji Matsumoto and Rashmi Prasad, editors, *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL https://aclanthology.org/C16-1079.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings, 2017. URL `http://hdl.handle.net/11234/1-1989`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18 (5):602–610, 2005. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2005.06.042. URL `https://www.sciencedirect.com/science/article/pii/S0893608005001206`. IJCNN 2005.

Roman Grundkiewicz and Marcin Junczys-Dowmunt. The wiked error corpus: A corpus of corrective wikipedia edits and its application to grammatical error correction. In Adam Przepiórkowski and Maciej Ogrodniczuk, editors, *Advances in Natural Language Processing*, pages 478–490, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10888-9. URL `http://emjotde.github.io/publications/pdf/mjd.poltal2014.draft.pdf`.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In Helen Yannakoudakis, Ekaterina Kochmar, Claudia Leacock, Nitin Madnani, Ildikó Pilán, and Torsten Zesch, editors, *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4427. URL `https://aclanthology.org/W19-4427`.

Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. Facebook data for sentiment analysis, 2013. URL `http://hdl.handle.net/11858/00-097C-0000-0022-FE82-7`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Jan Hajič, Jaroslava Hlaváčová, Marie Mikulová, Milan Straka, and Barbora Štěpánková. MorfFlex CZ 2.0, 2020. URL `http://hdl.handle.net/11234/1-3186`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Dana Hlaváčková, Hana Žižková, Klára Dvořáková, and Martkéta Pravdová. Developing online czech proofreader tool: Achievements, limitations and pitfalls. *Bohemistyka*, 22(1):122–134, mar. 2022. doi: 10.14746/bo.2022.1.7. URL https://pressto.amu.edu.pl/index.php/bo/article/view/31758.

Satoru Katsumata and Mamoru Komachi. Stronger baselines for grammatical error correction using a pretrained encoder-decoder model. In Kam-Fai Wong, Kevin Knight, and Hua Wu, editors, *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 827–832, Suzhou, China, December 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.aacl-main.83.

Vojtěch Kovář. Partial grammar checking for czech using the set parser. In *International Conference on Text, Speech and Dialogue*, 2014. URL https://api.semanticscholar.org/CorpusID:36173250.

Vojtěch Kovář, Aleš Horák, and Miloš Jakubíček. Syntactic analysis using finite patterns: A new parsing system for czech. In *Human Language Technology. Challenges for Computer Science and Linguistics*, pages 161–171, Berlin/Heidelberg, 2011. Springer. ISBN 978-3-642-20094-6. URL http://dx.doi.org/10.1007/978-3-642-20095-3_15.

Vojtěch Kovář, Jakub Machura, Kristýna Zemková, and Michal Rott. Evaluation and improvements in punctuation detection for czech. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue 19th International Conference, TSD 2016 Brno, Czech Republic, September 12–16, 2016 Proceedings*, pages 287–294, Cham (CH), 2016. Springer. ISBN 978-3-319-45509-9. doi: http://dx.doi.org/10.1007/978-3-319-45510-5_33.

Michal Křen, Václav Cvrček, Tomáš Čapka, Anna Čermáková, Milena Hnátková, Lucie Chlumská, Tomáš Jelínek, Dominika Kováříková, Vladimír Petkevič, Pavel Procházka, Hana Skoumalová, Michal Škrabal, Petr Truneček, Pavel Vondřička, and Adrian Zasina. SYN v4: large corpus of written czech, 2016. URL http://hdl.handle.net/11234/1-1846. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

David Kubeša and Milan Straka. DaMuEL 1.0: A large multilingual dataset for entity linking, 2023. URL http://hdl.handle.net/11234/1-5047. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Jakub Machura. *Automatické doplňování a korekce interpunkce v češtině*. Disertační práce, Masarykova univerzita, Filozofická fakulta, 2022. URL `https://is.muni.cz/th/xcpso/`.

Jakub Náplava and Milan Straka. Grammatical error correction in low-resource scenarios. In Wei Xu, Alan Ritter, Tim Baldwin, and Afshin Rahimi, editors, *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 346–356, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5545. URL `https://aclanthology.org/D19-5545`.

Jakub Náplava, Milan Straka, Pavel Straňák, and Jan Hajič. Diacritics restoration using neural networks. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL `https://aclanthology.org/L18-1247`.

Jakub Náplava, Milan Straka, Jana Straková, and Alexandr Rosen. Czech grammar error correction with a large and diverse corpus. *Transactions of the Association for Computational Linguistics*, 10:452–467, 2022. doi: 10.1162/tacl_a_00470. URL `https://aclanthology.org/2022.tacl-1.26`.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. The CoNLL-2014 shared task on grammatical error correction. In Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant, editors, *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-1701. URL `https://aclanthology.org/W14-1701`.

Jakub Náplava. Natural language correction with focus on czech, 2022.

Jakub Náplava, Milan Straka, and Jana Straková. Diacritics restoration using bert with analysis on czech language. *Prague Bulletin of Mathematical Linguistics*, 116(1):27–42, April 2021. ISSN 0032-6585. doi: 10.14712/00326585.013. URL `http://dx.doi.org/10.14712/00326585.013`.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. GECToR – grammatical error correction: Tag, not rewrite. In Jill Burstein, Ekaterina Kochmar, Claudia Leacock, Nitin Madnani, Ildikó Pilán, Helen Yannakoudakis, and Torsten Zesch, editors, *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.bea-1.16. URL `https://aclanthology.org/2020.bea-1.16`.

Anna Plocková. Automatické opravy chyb v psaní mně, mě. Diplomová práce, Masarykova univerzita, Filozofická fakulta, 2019. URL `https://is.muni.cz/th/cqhkg/`.

Michal Richter, Pavel Straňák, and Alexandr Rosen. Korektor – a system for contextual spell-checking and diacritics completion. In Martin Kay and Christian Boitet, editors, *Proceedings of COLING 2012: Posters*, pages 1019–1028, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL `https://aclanthology.org/C12-2099`.

Alexandr Rosen. Building and using corpora of non-native czech. In Brona Brejová, editor, *Proceedings of the 16th ITAT Conference Information Technologies - Applications and Theory, Tatranské Matliare, Slovakia, September 15-19, 2016*, volume 1649 of *CEUR Workshop Proceedings*, pages 80–87. CEUR-WS.org, 2016. URL `https://ceur-ws.org/Vol-1649/80.pdf`.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. A simple recipe for multilingual grammatical error correction. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 702–707, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-short.89. URL `https://aclanthology.org/2021.acl-short.89`.

Pavel Rychlý. Czaccent - simple tool for restoring accents in czech texts. In *RASLAN*, 2012. URL `https://api.semanticscholar.org/CorpusID:3595966`.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost, 2018.

Milan Straka, Jakub Náplava, and Jana Straková. Character transformations for non-autoregressive GEC tagging. In Wei Xu, Alan Ritter, Tim Baldwin, and Afshin Rahimi, editors, *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 417–422, Online, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.wnut-1.46. URL `https://aclanthology.org/2021.wnut-1.46`.

Jana Straková, Milan Straka, and Jan Hajič. Open-source tools for morphology, lemmatization, POS tagging and named entity recognition. In Kalina Bontcheva and Jingbo Zhu, editors, *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-5003. URL `https://aclanthology.org/P14-5003`.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. Tense and aspect error correction for ESL learners using global context. In Haizhou Li, Chin-Yew Lin, Miles Osborne, Gary Geunbae Lee, and Jong C. Park, editors, *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 198–202, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `https://aclanthology.org/P12-2039`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Jonáš Vidra, Zdeněk Žabokrtský, Lukáš Kyjánek, Magda Ševčíková, Šárka Dohnalová, Emil Svoboda, and Jan Bodnár. DeriNet 2.1, 2021. URL `http://hdl.handle.net/11234/1-3765`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Klára Vostřelová. Automatická detekce chyb v psaní velkých písmen v češtině. Diplomová práce, Masarykova univerzita, Filozofická fakulta, 2019. URL `https://is.muni.cz/th/r89h9/`.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL `https://aclanthology.org/2021.naacl-main.41`.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. A new dataset and method for automatically grading ESOL texts. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 180–189, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `https://aclanthology.org/P11-1019`.

Zheng Yuan and Ted Briscoe. Grammatical error correction using neural machine translation. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1042. URL `https://aclanthology.org/N16-1042`.

Zheng Yuan and Mariano Felice. Constrained grammatical error correction using statistical machine translation. In Hwee Tou Ng, Joel Tetreault, Siew Mei Wu, Yuanbin Wu, and Christian Hadiwinoto, editors, *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 52–61, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `https://aclanthology.org/W13-3607`.

Zdeněk Žabokrtský, Magda Ševčíková, Milan Straka, Jonáš Vidra, and Adéla Limburská. Merging data resources for inflectional and derivational morphology in Czech. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1307–1314, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://aclanthology.org/L16-1208`.

Karel Šebesta. Korpusy češtiny a osvojování jazyka. *Studie z aplikované lingvistiky [Studies in Applied Linguistics]*, pages 11–33, 2010. URL `https://sites.ff.cuni.cz/studiezaplikovanelingvistiky/wp-content/uploads/sites/19/2016/03/karel_sebesta_11-33.pdf`.

Dana Šmatová. Nejčastější pravopisné chyby žáků na 2. stupni ZŠ a jak jim předcházet. Bakalářská práce, Západočeská univerzita v Plzni, 2015. URL `http://hdl.handle.net/11025/19677`.

Jan Švec, Jan Lehečka, Luboš Šmídl, and Pavel Ircing. Automatic correction of i/y spelling in czech asr output. In *TDS*, pages 321–330, Cham, 2020. Springer

International Publishing. ISBN 9783030583224. URL http://hdl.handle.net/11025/43118.