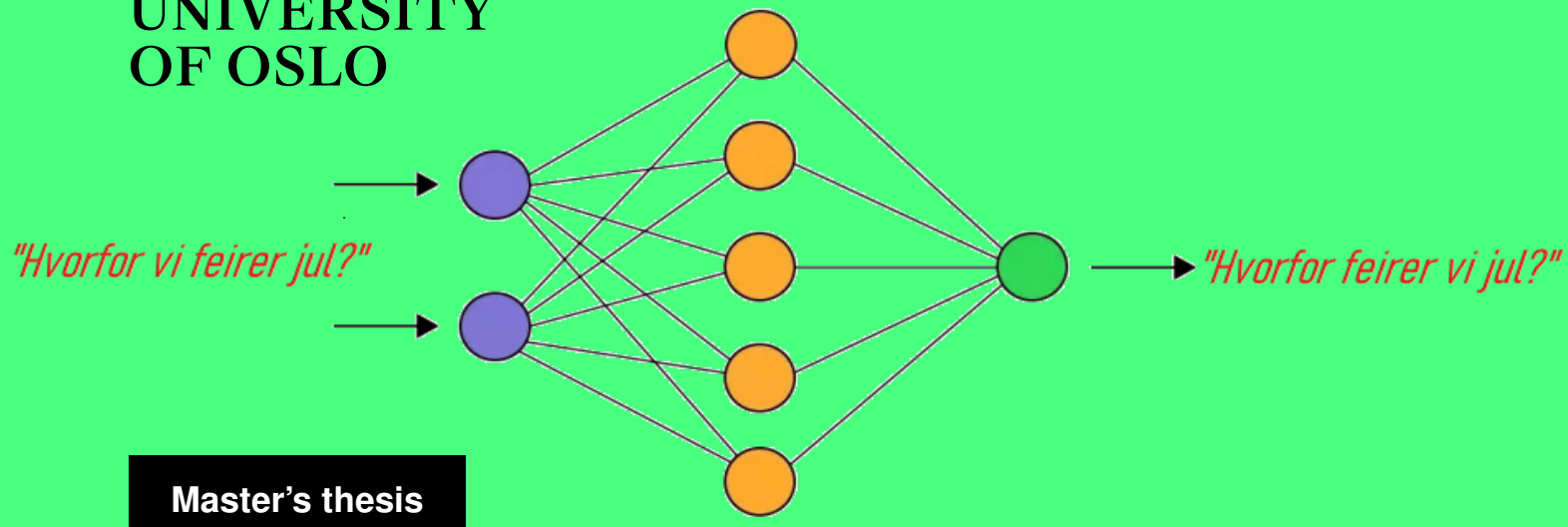
UNIVERSITY
OF OSLO



Master's thesis

Grammatical Error Correction with byte-level language models

Fine-tuning models for English and Norwegian

Matias Jentoft

Language Technology
60 ECTS study points

Department of Informatics
Faculty of Mathematics and Natural Sciences

Spring 2023

Abstract

Grammatical Error Correction (GEC) is the task of automatically correcting errors in human language on a sentence level or higher. Good performance on this task can have many real-world applications for foreign language learners, children, and people with various language impairments. In this thesis we explore how different representations of language affect the performance of GEC systems for English and Norwegian. We compare sequence-to-sequence language models with byte-level (`byt5`) and subword-level (`t5/nort5/mt5`) language representation. We show that byte-level representation of model input is just as good for this kind of tasks as subword-level representation, and for Norwegian it is actually better. We look at how the different levels of representation affect performance on specific error types that may occur in natural text. A special focus is put on how byte-level representation handles noisy text, i.e. text that contains spelling errors that might not have occurred in the training data of the models.

We release the first language models for grammatical error correction in Norwegian. This has been possible with a large annotated second language learners corpus: Norsk Andrespråkskorpus (ASK). We modify the corpus into a parallel corpus, and use it for fine-tuning pre-trained models of the `t5`-family. Our best system, which is byte-level based and trained on multilingual data, achieves an $F_{0.5}$ score of 0.581 on our test set, and beats the subword-level Norwegian-trained `nort5`.

Acknowledgements

My supervisors deserve a big thank you for making this project happen. David Samuel came up with the idea to work on Grammatical Error Correction with byte-level representation, and he figured out that there was a Norwegian dataset suited for this purpose. He has been patient and inspiring with his infinite scientific and technical knowledge. My other supervisor Andrey Kutuzov has given me motivation and inspiration to keep working and staying curious about new publications. It has been a great pleasure to work with them both!

I also want to thank my partner Katharina and our cat Tassen for letting me go through with getting a degree at the university. I have heard absolutely no complaints, even when I've been at uni the whole night, or my self-esteem has been at rock bottom. I want them to know how much I appreciate the feeling of freedom it gives me to know that I can mold my own destiny. Lastly, I want to thank the whole gang at lesesal 4126. Sharing the master thesis boat with them I have realized how important social interaction is for my style of learning and working. Also that always having communal snacks available increases life quality.

Matias Jentoft Oslo, 14.05.23

Contents

1 Introduction **1**

2 Background **3**
2.1 Grammatical Error Correction 3
2.1.1 Evaluation of GEC 4
2.1.2 Three periods of development in GEC 5
2.1.3 Previous results on shared task benchmarks 9
2.2 Language representation in NLP 9
2.2.1 Word level language representation 10
2.2.2 Subword language representation 11
2.2.3 Byte and character level language representation . . 12
2.2.4 Language representation for GEC 13
2.3 Pre-trained GEC models 14
2.3.1 Transformers 14
2.3.2 Transfer learning 16
2.3.3 t5 16
2.3.4 nort5 17
2.3.5 mt5 18
2.3.6 byt5 18
2.4 GEC for Norwegian 19

3 Datasets **20**
3.1 General requirements 20
3.2 Error annotation 21
3.2.1 Annotation challenges 22
3.3 Data sparsity 23
3.4 The datasets we use 24
3.4.1 CLC-FCE 25
3.4.2 NAIST 26
3.4.3 ASK 27
3.5 Datasets in other languages 30
3.6 Datasets for evaluation 30
3.7 NoCoLA - another application for a parallel corpus 31
3.7.1 Language model evaluation 31
3.7.2 NoCoLA 31
3.7.3 Comparing NoCoLA to GEC parallel datasets 32

4 Evaluation **34**
4.1 Evaluating seq2seq . . . 34
4.2 Evaluating GEC . . . 35
4.2.1 Level of evaluation . . . 35
4.3 Metrics . . . 36
4.3.1 $F_{0.5}$ . . . 36
4.3.2 Other reference based metrics . . . 37
4.3.3 Reference-less metrics . . . 37
4.4 Scorers . . . 38
4.4.1 MaxMatch . . . 38
4.4.2 ERRANT . . . 38
4.4.3 ERRANT for Norwegian . . . 39
4.4.4 ERRANT for other languages . . . 40
4.4.5 ASKEVAL . . . 40
4.5 Conclusion . . . 40

5 Experiments and Results **42**
5.1 Methodology and general considerations . . . 42
5.1.1 Pretrained models . . . 42
5.1.2 Evaluation data . . . 43
5.1.3 Metrics and reported numbers . . . 43
5.1.4 Software and hardware . . . 43
5.1.5 Model hyperparameters . . . 43
5.2 English . . . 45
5.2.1 Round 0 - `t5` without fine-tuning . . . 45
5.2.2 Round 1 - basic `t5` . . . 46
5.2.3 Round 2 - gradient accumulation . . . 47
5.2.4 Round 3 - training data modifications . . . 49
5.2.5 Round 4 - `mt5` and `byt5` . . . 50
5.2.6 Round 5 - testing on test data . . . 50
5.3 Norwegian . . . 51
5.3.1 Evaluation for Norwegian . . . 53
5.3.2 On the datasets . . . 53
5.3.3 Round 6 - Norwegian ASK - development . . . 53
5.3.4 Round 7 - Norwegian ASK - test . . . 55
5.4 Summary . . . 56

6 Analysis **57**
6.1 English - fine-grained-ERRANT . . . 57
6.1.1 Conclusions . . . 61
6.2 Norwegian - fine-grained-ASKEVAL . . . 62
6.2.1 Caveats when using ASKEVAL . . . 62
6.2.2 Result discussion . . . 62
6.3 Norwegian - effect of data augmentation . . . 65

7 Conclusion **69**
7.1 Byte level representation on English GEC . . . 69
7.2 GEC for Norwegian . . . 70

8 Suggestions for future work **71**
8.1 More GEC training data for Norwegian 71
8.2 GEC evaluation for Norwegian 72
8.3 Language specific byte-level models 72
8.4 GEC and society 72

Appendix **81**
8.5 Error wise results round 5 81

List of Figures

2.1 Illustration: the transformer architecture 15

3.1 Illustration: source and target for GEC 20
3.2 Example: the XML-format 21
3.3 Example: M2-format 21
3.4 ASK: error type distribution 28

4.1 Simple example: error span evaluation 36
4.2 Examples: ERRANT for Norwegian 39

5.1 Timeline: `t5` versus `byt5`, round5 52

6.1 Bar-chart: error-wise comparison of models, granularity levels 1 and 2 together 58
6.2 Bar-chart: error-wise comparison of models, granularity level 3: all three plots 59
6.3 Bar-chart: error-wise comparison of models (Norwegian) . . 63
6.4 Tables for each error type in ASK 64
6.5 ASKEVAL: all results in tables 65
6.6 Bar-chart: effect of data augmentation (Norwegian) 68

8.1 Evaluation: ERRANT on `t5` 83
8.2 Evaluation: ERRANT on `mt5` 85
8.3 Evaluation: ERRANT on `byt5` 87

List of Tables

2.1 Results: shared tasks on GEC 2014/2019 9

3.1 Overview of GEC datasets used in the thesis 25
3.2 FCE: error-type explanations 25
3.3 Explanations: ASK error types 29
3.4 Sizes of NoCoLA datasets 32

5.1 Default generation hyperparameters of `t5` 45
5.2 Results: round 0 45
5.3 Example predictions: zero-shot example with `t5` 45
5.4 Results: round 1 46
5.5 Example predictions: round 1 47
5.6 Sizes of the models in the `t5`-family 48
5.7 Results: round 2 48
5.8 Results: round 3 49
5.9 Results: round 4 50
5.10 Results: round 5 51
5.11 ASK datasets: sizes 53
5.12 Results: round 6 (Norwegian) 54
5.13 Results: `nort5` adjusted learning rates 54
5.14 Results: round 7 (Norwegian) 55

6.1 ERRANT error-types abbreviations 57
6.2 Examples: `nort5` versus `byt5` in round 7 66

Chapter 1

Introduction

Natural Language Processing, or just NLP, is a field within informatics that concerns itself with automating tasks related to human natural language. A familiar automated task for many users is that of spell-checking, where user generated text is the input, and the system identifies wrongly spelled words and suggests corrections. For the same kind of task, on a higher level, a sentence that possibly contains language errors can be the input, and a correctly spelled version of the same sentence can be the output. In practice this can mean suggesting the changes to the user, or changing the text automatically. This task is called Grammatical Error Correction (GEC) in NLP. Today it is usually performed on individual sentences, but it could also be performed on a paragraph or document level. A system that solves this task does not only correct misspelled words, but also tackles grammatical and semantic errors, like those related to word order or agreement between words.

There are at least three groups of people that could benefit from the use of GEC: children learners, foreign language learners, and people of all ages with language impairments. Society as a whole can also benefit from GEC availability in saving costs from correcting texts written by these groups. An additional social benefit is that all citizens would be able to participate in the public discourse, regardless of their linguistic background.

Our contribution in GEC research is two-fold. First, we have experimented with different levels (byte and subword) of language representation for GEC on well-known English datasets. Second, we have trained the first GEC models for Norwegian with modern neural network architectures.

GEC is an NLP task that can be cast as a sequence-to-sequence (seq2seq) task (Omelianchuk et al., 2020), where the original sentences are the input and the corrected sentences are the output. This is similar to how modern neural machine translation (NMT) works, except that GEC is done within one single language. In this analogy the possibly erroneous sentence is in the source language for translation, and the corrected version is the target language for translation. From a training perspective, given the appropriate neural network architecture, the system only needs pairs of original and corrected sentences as input. After training, the system can automatically

perform corrections on unseen, possibly faulty, input sentences. For our experiments, we have used pre-trained seq2seq models in the `t5` family (Raffel et al., 2020) when doing fine-tuning on the task of GEC. The original English `t5` model, and its multilingual counterpart `mt5` (Xue et al., 2021), represent sentences with subword level embeddings. The family also includes a byte-level model, `byt5` (Xue et al., 2022). By comparing the subword- and byte-level versions we have the opportunity to study how the difference in text representation affects performance on GEC. One hypothesis is that byte-level language models are good at handling so-called noisy data, meaning surprising and misspelled words not seen by the system before. We ask whether different representation levels influence overall GEC performance, and also performance on specific error types.

Our second contribution is to introduce the first models trained for conducting GEC in Norwegian, a North Germanic language with about 5 million native speakers, considered a medium resource language in the NLP community. We have achieved this by utilizing the Norwegian corpus of second language learners (Norsk andrespråkskorpus: ASK; Tenfjord et al., 2006). We compare the use of pre-trained models trained on multilingual data, and models only trained on Norwegian data. Also for Norwegian we ask how byte-level representation affects GEC system performance. Can a multilingual, byte-level model, be just as good for GEC as a model specifically pre-trained on Norwegian data? Lastly we explore some possibilities for maximising the utility of limited training data, called data augmentation.

In chapter 2, we give an overview of previous work on grammatical error correction, as well as a description of the models we are using to conduct our experiments. Chapter 3 introduces the datasets that we are using, both for English and Norwegian. For the Norwegian dataset we also describe some other uses for the parallel corpus we have created. Evaluating the quality of GEC predictions is not a straightforward exercise, and chapter 4 is dedicated to that subject. The main weight of the thesis is the experiments and analysis in chapters 5 and 6, where we among other things test the brand new `nort5` (Samuel et al., 2023) model, and try to answer the research questions outlined above. Lastly we summarise the thesis and suggest some possible future lines of inquiry in chapters 7 and 8.

We release our code and Norwegian GEC dataset[1], and the Norwegian GEC model checkpoints [2] together with the thesis.

[1] https://github.com/matias-jjj/matias_master
[2] https://huggingface.co/MatiasJ

Chapter 2

Background

In this chapter we give an overview on background work that is relevant for our experiments. Section 2.1 is about how grammatical error correction is defined as a task, and how the progress on solving it has been through the years. In section 2.2 we discuss what language representation alternatives there are in NLP. Section 2.3 describes the architectures of the neural network models that can be used for automating GEC development. The last section summarises the state of GEC for Norwegian language.

The main purpose of the chapter is to motivate our experiments, as well as to give the necessary background info needed to understand the research questions from chapter 1 and the experiments in chapter 5.

2.1 Grammatical Error Correction

The goal of a GEC system is to automatically transform all sentences, faulty or not, into grammatically acceptable sentences. This task is traditionally performed by human experts, such as teachers, proofreaders, or other proficient language users. They read the original texts and either mark mistakes with a red pen or transform the mistakes so that the result is correct. This can be a very resource consuming task. There are some benefits to this human approach in a language learner situation, because learning, especially for children, can be a very social activity. In this thesis, that aspect is not in focus, and the focus is primarily on systems that create new, corrected sentences, without any annotation or pedagogical considerations. Methods for how to best make use of automatic GEC in society is left for future work, requiring knowledge from various other scientific fields.

A note on the term Grammatical Error Correction itself. Most datasets used for training and evaluation consist of natural text by language learners who generate errors on all levels, all the way from from orthographical to pragmatical. Thereby the corrections asked for are not exclusively "grammatical" in nature. The nature of the task is largely defined by what errors and corrections are included in each dataset, and particularly what errors are included in the test sets of the shared tasks ConLL14 (Ng et al., 2014) and BEA-19 (Bryant et al., 2019) (Bryant et al., 2023). As suggested in the aforementioned survey, perhaps the term Language Error Correction

```
# Replacement (eng):
    I look forward to hear some news from you!
    I look forward to hearing some news from you!
# Replacement (nor):
    Da jeg kom hit, begunte jeg på norskkurs.
    Da jeg kom hit, begynte jeg på norskkurs.
# Omission (eng):
    But it was most important place in the world.
    But it was the most important place in the world.
# Omission (nor):
    Først gikk jeg på norskkurs i Oslo 3 måneder.
    Først gikk jeg på norskkurs i Oslo i 3 måneder.
# Insertion (eng):
    I was in her room and helping her.
    I was in her room helping her.
# Insertion (nor):
    Hun fortalte alt om sitt liv, og jeg om mitt liv.
    Hun fortalte alt om sitt liv, og jeg om mitt.
```

Listing 1: Examples of the three broadest categories of errors in both English (CLC-FCE) and Norwegian (ASK). The original, or source, sentence is first, and the corrected, target, version second

(LEC) would be more descriptive. But GEC is the term universally used in the field, and we are following that tradition in this thesis.

So what exactly do we want a GEC system to do? The changes required from a GEC model can be broadly classified into three categories. The first is *replacement*, where a word or phrase should be substituted by another. The second is *omission* errors, where a word or phrase should be added. The third is *insertion* errors, where a unnecessary word should be removed. Some examples of the three types of actions are listed in table 1.

2.1.1 Evaluation of GEC

A more thorough discussion on the metrics and scorers we use for GEC in this thesis can be found in chapter 4. Here comes a short explanation of what we need to know about evaluation before we delve into the history of GEC.

To be able to know how well a system performs a task, one needs a good strategy for automatically evaluating the quality of the output. Correction-span based $F_{0.5}$ is the standard metric to evaluate GEC. The minimum score is 0.0, and the maximum is 1.0. The methodology relies on a source sentence, a target sentence, and the prediction from the model. Identical corrections in prediction and target are rewarded. The parts of the sentences that are identical in all three are ignored during evaluation, and comparison and evaluation is only done on the parts that are different. A scorer, for example MaxMatch (Dahlmeier & Ng, 2012) or ERRANT (Bryant et al., 2017) is

needed to automatically detect the spans and calculate the score.

2.1.2 Three periods of development in GEC

Our primary sources in this section are the three survey papers by Y. Wang et al. (2021), Bryant et al. (2023), and M. R. Qorib and Ng (2022). We follow Y. Wang et al. (2021) in dividing the development within the GEC field into three phases. Following this taxonomy, the upcoming subsections present how the approach towards the task has developed in the technological era, and how this specific task relates to the rest of NLP research — specifically machine translation.

Early

As in many other subfields of NLP, deductive thinking dominated the early years of GEC. Strict rules are defined by human experts, and these rules can be applied to new text so as to generate corrected text, or suggestions for corrections. Examples of tools created with this strategy, mentioned in Y. Wang et al. (2021), are the Language Tool (Naber, 2003) and the ESL Assistant (Gamon et al., 2009). The kind of rules seen in such systems can be formulated on the form: if the word "thanx" appears in the input, change it to "thanks", or if next word starts with a vowel, change "a" to "an".

Rule based models are not very flexible, and are limited to the mistakes that the expert rule-makers remember to include explicitly. Such rules are also harder to device for very context-dependent error-types, as it can be hard to generalize rules for them. It also takes a lot of work to create, and the rules have to be updated when problems are detected.

In these earlier stages of development, the GEC task was sometimes split up into subtasks. The task was not necessarily to do a correction of *all* mistakes, like it is in later years of development in the field, but perhaps just focus on correcting wrong prepositions. One strategy for correcting particular errors, or error types, is to use classifiers. The input is defined as a feature vector, and the prediction is a class from a closed group of classes. Then one needs to have a separate classifier for each error type. Keeping in mind the vast array of possible errors language learners might make, the list of different classifiers quickly becomes very big.

Middle - statistical

When access to big data and storage capacity has grown, data-driven, induction-based systems have gradually taken over for rule-based systems. This shift of attention simmered into GEC research through machine translation (MT), as GEC can also be cast as a translation task from erroneous to correct sentences. Rules can be indirectly inferred from the patterns in texts already written and annotated by humans. Instead of rules making up the model, the model is created from patterns seen during training on the parallel sentences of the corpus.

In the first phases of development, the application of this data-driven strategy to "real" translation and GEC is called statistical machine translation, or SMT. The important consideration is that the input and output sequences can be of arbitrary length, and it is not a word-by-word translation. There are many reasons why this is beneficial, for example a.) words missing in the original text, b.) there are superfluous words in the original sentence or c.) large parts of the sentence have to be rewritten. This becomes evident in (human) translation and machine translation as well, as the translations of the same book can have radically different page counts than the original work.

SMT is inspired by the noisy channel model, which again is based on Bayes rule (Bryant et al., 2023). An SMT model roughly creates a probability for possible target sentences given source sentences, and chooses the output sentence with the highest probability. Mathematically this means to maximise the probability $p(y|x)$, where y is the target sequence, and x is the input sequence. In addition, a language model is often used for the target language to make sure the output sentence is statistically probable, and thereby natural and fluent.

The paper by Brockett et al. (2006) is a seminal study of SMT-techniques for GEC. It focused only on a subset of the general GEC task: on correcting countability errors of mass nouns (Y. Wang et al., 2021). Their system was able to correct 61.81 percent of mistakes in a set of naturally-occurring examples of mass noun errors. At the height of SMT development, SMT-models achieved first and third place in CoNLL-2014 shared task (Ng et al., 2014).

Next we describe in some detail how a GEC-SMT might work. In Behera and Bhattacharyya (2013) a hierarchical phrase-based SMT is presented. As is typical for the pre-neural era, the training consists of an extensive pipeline of processes. The tokens are pre-processed with regards to case, and the words of the original and corrected sentences are aligned. Then the system learns which phrase-level transitions are needed to go from source to target sentence, and these transitions become rules. Now the rules can be applied to the source sentences of the development data, and the result of this is compared to the gold data. In this way the ordering of the rules can be tuned for the task at hand. They report a BLEU-score of 0.774 on the NUCLE (National University of Singapore Corpus of Learner English) dataset (Dahlmeier et al., 2013). BLEU, a metric common in machine translation, is further described in chapter 4. Shortly, it must be seen as a less strict metric than the $F_{0.5}$ metric we use in this thesis.

Foreshadowing the use of character level language models in later neural systems, there is already experimentation with using elements of this thinking in SMTs applied to GEC. Chollampatt and Ng (2017) uses a character level SMT for spellchecking, which according to the writers improves the performance. This component is used only for generating candidate corrections for misspelled words. The candidate that best fits the context is chosen. It seems that this is a separate component only meant to take unknown words and outputting a collection of correct candidates, and is not in use for representing known words.

Late: neural

There has been a neural revolution in NLP in the last 5–10 years. This means a new type of models called neural networks that transform input to output without the need for explicit feature selection. The architecture consisting of multiple layers of neurons, or nodes, where linear transformations are performed on the data. Between these transformations, non-linear transformations are performed on the output values of the nodes, or on the input feature values in the case of the first layer. The non-linearity allows such networks to handle data which is not linearly separable in a multidimensional vector space. The linear transformations consist of parameters, or weights, that control how different aspects of the input affect the output. The central part is that these parameters can be automatically trained on training data with an algorithm called back-propagation, which automatically improves the predictive abilities of the model by adjusting the weights according to current predictions. Again, since a lot of the strategies used for GEC have inspiration from the machine translation community, the name neural machine translation, NMT, is used in the field of GEC as well.

Some techniques for performing a seq2seq transformation are recurrent neural networks (RNNs) and convolutional neural networks (CNNs), which both "read" a sequence sequentially; either one way (left to right in European languages), or both ways. The latter gives a model the ability to use words from the future to make predictions at the current time-step. Fo long term dependencies, which are grammatical or semantic relationships between words that are not adjacent in sentences, gated units are used in RNNs. Two of the most widely used gated units are Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho, van Merriënboer, Gulcehre, et al., 2014).

The historical survey paper by Y. Wang et al. (2021) describes the prime advantages of these new models to be that such models map directly from input to data, without need to manually define features to describe the input. One can say that the important features are automatically learned in the early "hidden" layers of the models. Another advantage is that neural models can better correct mistakes on a phrasal and sentence level, and they are also better at generalizing, by correcting mistakes not previously seen in the training data.

NMT based models in GEC were first applied in Yuan and Briscoe (2016). They achieved an $F_{0.5}$ score of 39.90 on the CoNLL-2014 shared task test set (Ng et al., 2014). The shared tasks created for GEC are presented in section 2.1.3. The score is significantly lower than the state-of-the-art models of today, but was competitive at the time. Their architecture consists of a bidirectional recurrent neural network (RNN) transforming the sentence into a uniformly sized context vector. This part of a translation-type model is called the encoder. The decoder, which generates the corrected sentence from the context vector and the original sentence, is attention-based. Attention is a mechanism that allows the model to focus on the relevant parts of the source sentence at any given timestep when predicting the next word. To solve the problem of unknown words, for example misspelled words

which are not in the vocabulary of the language, they apply an unsupervised alignment model called GIZA++, which creates an alignment between the predicted unknown words and their counterparts in the source sentence. Then these two words can be used to build a word-level translation-model, which is much simpler, and is treated as a separate task.

Xie et al. (2016) were the first to apply a character-level elements in their GEC system. Their architecture consists of an advanced version of RNNs called Gated Recurrent Units GRUs (Cho, van Merriënboer, Bahdanau, et al., 2014) in both encoder and decoder, with an attention mechanism that binds each character generation time-step to the different parts of the input. To surpass problems with out-of-vocabulary-words (OOV), like web addresses and misspellings, the system operates on character level instead of on token level. To save computing costs from doing calculations at each time step, one for each character in both encoder and decoder, for several layers of a the RNN, they construct a pyramid scheme where the higher layers are smaller than the lower ones. Their system achieves an $F_{0.5}$ score of 40.56 CoNLL 2014 test set. This is still significantly below the state-of-the-art results achieved in later years. However, the work of Xie et al. (2016) serves as important background and inspiration for the experiments in this thesis.

Attention, introduced in Bahdanau et al. (2014), is a mechanism which can be included in an RNN. At each prediction step in the decoder, the model is allowed to pay attention to the elements of the input to a varying degree. This mechanism produced better results in machine translation, and was introduced to GEC in Yuan and Briscoe (2016).

This attention mechanism would later become even more central in GEC models. After RNNs and its derivatives like LSTM and GRU, transformer-based models (Vaswani et al., 2017) have taken over as the most popular and standard tool for performing sequence-to-sequence tasks. Transformers, just as RNNs, have an encoder-decoder architecture. The main difference is that each token in the encoder and decoder are not processed sequentially based on the previous tokens, but rather processed all at once. This is done solely with the help of the attention mechanism. The prime advantage of using transformers instead of RNNs for the encoding and decoding in a seq2seq type architecture, is that operations can be conducted in parallel to a higher degree, and thereby training speed is increased. Conceptually, the use of transformers is very different from RNNs and human language processing, as a sequence is seen as a whole, and the processing of a unit is not dependent on the processing of the previous context. This is what makes parallel processing possible. The central feature behind this ability is called self-attention.

An example of GEC by using transformers is presented in Grundkiewicz et al. (2019). The model is an seq2seq architecture, just like in the RNNs mentioned above. But the encoder and decoder are transformer self-attention blocks instead of RNNs. The output of each cell in each block is not calculated based on the previous time step, but instead the calculation is based on the content of all the other elements in the sequence and trained weights. In addition to implementing a transformer architecture, this paper focuses on unsupervised synthetic training data generation, which is a

subject we get back to in chapter 3.

2.1.3 Previous results on shared task benchmarks

Authors	System type	$F_{0.5}$ score
2014		
Yuan and Briscoe (2016)	rnn	39.39
M. Qorib et al. (2022)	ensemble, including t5	69.51
Rothe et al. (2021)	tf	68.87
Grundkiewicz and Junczys-Dowmunt (2018)	smt + rnn	56.25
2019		
Grundkiewicz et al. (2019)	tf	69.47
M. Qorib et al. (2022)	ensemble, including t5	79.90
Rothe et al. (2021)	tf	75.88
Li et al. (2019)	ensemble/rnn	66.78
Tarnavskyi et al. (2022)	tf	76.05

Table 2.1: Previous results on the 2014 (Ng et al., 2014) and 2019 (Bryant et al., 2019) shared tasks. In the authors column is a reference to the paper which accompanies each system. tf: transformer, rnn: recurrent neural network, smt: statistical machine translation

Quite many shared tasks have been created for English GEC. This means at its core that a parallel GEC-corpus is held hidden. Participants create their own systems which are then tested and evaluated on this hidden data. Other rules might also apply, like the data one is allowed to train the systems on. In table 2.1 we show a selection of results on the shared tasks of CoNLL-2014 and BEA-2019. Some of the results are not listed in the official listings of the shared tasks, as the tests on the test set have been done after the official deadlines of the task, or the results have not been officially presented on conferences yet.

There have been efforts to stimulate GEC development through shared tasks for other languages as well. One example is the Second Ukrainian Natural Language Processing Workshops (UNLP) shared task for Ukrainian GEC of 2023 [1].

2.2 Language representation in NLP

For any neural architecture used for NLP one needs to have a useful, numerical representation of the training data. This is true for all modern NLP-tasks where calculations are to be performed. The usefulness of such a representation is an abstract concept. We can loosely describe usefulness as how much semantic information the representation contains. For GEC, the language representation is relevant in the way a sequence is represented when it is used as input for training, and the way the possible output

[1] https://unlp.org.ua/shared-task/

vocabulary is represented numerically. For efficiency and environmental reasons, creating such representations is often a shared effort, in the sense that the same base model is used by several teams for different tasks.

A quick note on the term "language models". The term might sound confusing to some, as are not all models developed and used in NLP related to language? The etymology of the word comes from the field of automatic speech processing, where processing is sometimes done in two steps: first the physical sound signal is processed into a sample of its most likely language sounds. Then a model that gives probabilities to different sequences decides the output. This has been called a language model. In NLP this term has come to mean two different things: the first and strictest is a model that has the ability to predict the most likely next word of a sequence, or to similarly predict what a missing word in a phrase could be. Some examples of such models are `BERT` (Devlin et al., 2019), `ELMO` (Peters et al., 2018) and `GPT-2` (Winata et al., 2021). Second, it can be a term describing all kinds of models within NLP that have language as output, including seq2seq models.

Now we delve into the subject that at the core of this thesis: the different ways and levels of representing language in NLP systems.

2.2.1 Word level language representation

One traditional, and arguably intuitive, way to represent textual data numerically is that each *word* is represented as a numerical feature. The process of turning continuous text into smaller chunks is called tokenization, and in the case of word level tokenization, the split is done on white-spaces. With this scheme a token corresponds to what is considered a *word* in layman terms.

There are variations within word-level representation. The first and second variations is to represent words as word types. In its simplest form this means that a word is represented a one-hot vector which is a unique identifier for each word type. The vectors generated from such tokens is the size of the vocabulary, and consists of only zeros (0) except for one one (1) identifying the target word. This has several problems: the vectors are huge and slow to operate on, and there is no relationship between words. The identifier for the English word "cat" is as different from "kitten" as it is from "advanced". These kind of representations are solely unique identifiers for a given string of characters.

If one is to build a language model based on these word representations, the probabilities for a given sequence is based only on the frequencies seen in the training data for exactly matching sequences, and the similarity in meaning between words is ignored.

The second more complex type-style variation is to represent words with a low-dimensional array of continuous values, called static word embeddings. This representation strategy requires training on training data. Static word embeddings contain semantic information about the tokens they represent, following the distributional hypothesis of lexical

semantics. The semantic information is not referential in the sense that the information points to aspects or objects in the world around us. Instead it is constructed from the context that the word appears in in the training data, in a closed, linguistic system. Words that appear in similar contexts get similar numerical representations, with the help of automated machine learning algorithms like skip-gram and CBOW (Mikolov et al., 2013). The context consists of the words surrounding the focus token in the texts. This means that the probabilities generated by a language model built on static word embeddings will reflect the fact that some words are more semantically similar than others, and the occurrence of "the kitten was sleeping" will increase the probability of "the cat is sleeping", given that the embeddings for kitten and cat are similar. But the representation is still of the type-style, since any given word type will always have exactly the same representation.

A third variation of word-level representation is contextual embeddings, which take context into account at inference time, and not only during training. With such embedding-systems, the English word "tree" will have a different representations in "a tree grows in the forest" and "she constructed a syntactic tree" at inference time. This is because the words around it, often referred to as the *context*, are different. This type of representation is often combined with subword tokenization, more on those in the next subsection.

Even if word level units might seem like a good solution for representation, several problems arise. The process of splitting up continuous text into tokens by whitespace leaves many decisions to be taken by the system designers, such as how to treat compound words like "menneskeape" in Norwegian, and how to treat multiword tokens like "don't" in English. Punctuation use also leads to decisions having to be made, such as with apostrophes and dashes.

Another issue with word-level tokens is that the creativity of language use means that even with a huge training data, there will always be unknown (out-of-vocabulary words, OOV) words to the system. This means words that have not been seen during system training. For GEC an additional problem arises, which is how to represent wrongly spelled words. All possible misspellings of a word will most likely be absent or rare in the training data. As misspellings are a common, and arguably essential, feature of the GEC datasets we are using, word-level embeddings might not be optimal for our purpose.

2.2.2 Subword language representation

All solutions to the representation-problem that involve units that are smaller than words but bigger than single characters, can be called subword representations. This describes a wide array of different solutions for tokenization, everything from 2 character n-grams to units the maximal size of known units from a given system vocabulary.

The subword strategy solves the OOV problem to a large degree, as there are much fewer types of subword units than there are word-level units in a language. But this comes at the cost of complexity. In the context of subword tokens, splitting text into word-sized units which was before

called *tokenization*, is now called *pre-tokenization*, see Mielke et al. (2021). The term *tokenization* is now describing the transition from word-sized tokens to smaller pieces. We end up with two pre-processing stages.

A solution with maximally sized units within a given vocabulary is exemplified in the pre-trained language model BERT (Devlin et al., 2019). Any word-level token which is not in the vocabulary gets represented by the minimal amount of constituent parts that *do* exist in the vocabulary of the pre-trained language model. For example the English word *monkeytoe* will be represented by two parts: *monkey* and *##toe*. BERT models have other differences from previous language models, in that each subword token representation is not static, but rather it is contextualised and varies according to the context it appears in, as described in section 2.2.1.

Three of our models, `t5`, `nort5` and `mt5`, correspond to this scheme of subword tokenization. More on these models later in this chapter.

2.2.3 Byte and character level language representation

In this thesis we try out another representation strategy called byte-level language modelling (BLLM), which is tried out (among others) in Xue et al. (2022). The strategy is to represent the text as an array of single bytes, which can be seen as the ultimate granularization of language representation. The tokenization step of such a model is to split up the text according to bytes, which is a task that does not leave a lot of choices to the model developers. The models that utilize this scheme can therefore be said to *operate on raw text*.

As mentioned, language models have traditionally operated on word- or subword-level. This old way of thinking demands that one performs so called tokenization or segmentation, where raw text is split into multi-character units. This task is different across language orthographies. When doing tokenization for a specific language there are several decisions which are up to the researchers to decide, like whether to split compound words into several units or not. The whole tokenization-step can be bypassed by operating on the bytes of the raw text directly. If one splits on bytes, the task is more or less trivial. A related approach, splitting on characters (CLLM) instead of bytes (BLLM), is also possible. If one splits on characters, some consideration has to be taken with regards to the language used, as some `Unicode` characters have longer byte representations than others. The resulting units are the tokens of the model, in the same sense that words or sub-words were tokens in the models previously mentioned. In addition to not requiring expensive and error-prone pre-processing, another advantage of CLLMs and BLLMs compared to static embedding methods and BERT is that they are robust to misspellings and other orthographically variation, as representations for words with only one letter difference will tend to be similar. There are some subword strategies that take the latter issue into consideration as well, like the contextualised embedding in ELMo (Peters et al., 2018).

The immediate problem facing byte- of character-based language models is that it demands a lot of computing power to operate on raw text (Xue

et al., 2022), since the "memory" has to go further back (or both left and right in the case of bidirectional RNNs and transformers). This is because a sequence of character-level representations will consist of more tokens than a word-level representation of the same sentence. Xue et al. (2022) argues that with modern architectures like transformers these costs can be surpassed. Another issue with BLLMs and CLLMs is that the parameters have low interpretability, because of the very low granularity of each representation.

CLLMs and BLLMs depend on character-level vocabularies (e.g. based on utf-8) and byte-level vocabularies (Xue et al., 2022). The former usually has many more vocabulary-related parameters than the latter, as it has an embedding for each character. For a given language that could be as few as 20-100, but a multilingual model should include a large proportion of the Unicode characters (144,000 as of February 11, 2022), and therefore can grow up to 47,000 vocabulary parameters in CharT5-36 (Xue et al., 2022). It will have some redundant embedding-parameters for any given language, as not all characters in a encoding scheme are present in all languages. Byte-level vocabularies, however, only have 256 parameters for the vocabulary, and can therefore "move" the saved parameters to other places in the model, keeping the same efficiency.

One of the models we use in this thesis, `byt5`, conforms to the BLLM scheme of language representation. More on that model later in this chapter. One of the objectives of this thesis is to study how these advantages and disadvantages of `byt5` affect the results on the task at hand, namely GEC. We do not use any CLLMs in this thesis.

Previous results from using BLLMs

Xue et al. (2022) creates a minimal-pair comparison of a subword-model (`mt5`) and a byte-level model (`byt5`), by testing the models on some downstream tasks. The general impression from their results is that `byt5` performs comparably, and even outperforms a subword-level model in some tasks, namely on noisy tasks and generative tasks. These task descriptions fit with our task at hand: GEC.

Architecture-wise the benefits can be summarised as simplicity and robustness, the disadvantages as additional computations both at training and inference time.

BLLMs have been successful in tasks like Lexical Normalization, see Samuel and Straka (2021), which was the winning entry of the shared task on the topic in 2021. GEC is a quite similar task, just on sentence level. We therefore think it could be fruitful to apply this new language modelling strategy to this task as well. As far as we know (March 2022) this has not been done yet, probably because the application of BLLMS and CLLMs is a very recent development in the field.

2.2.4 Language representation for GEC

If the GEC task is to be cast as a sequence-to-sequence (seq2seq) task, where then is the use of a particular language representation relevant? The

network architecture requires a sequence of numeric input-units for both the encoder (source sentences) and decoder (target sentences). Each of this units represent some aspect of the text, which can be either a byte, character, subword, or word. The important thing is that these units are indivisible, even if the text they are representing consists of multiple characters. The conversion from running text to such units, the tokenization, is where language representation level comes into play. It is worth noting that even with small units like bytes, the relationship between them gets represented at some point deeper in the network.

2.3 Pre-trained GEC models

In this section all the models we use in experiments are presented and described. As previously discussed we need models that take sequences of arbitrary length as input, and that also output sequences of arbitrary length. These kind of models are called sequence-to-sequence language models, or seq2seq, in NLP. Theoretically one could build such models from scratch. We have instead chosen to fine-tune pre-trained models for our experiments, instead of developing our own models with a seq2seq architecture. This means models that are first trained on general language modelling tasks on large text corpora, but are not yet fine-tuned for GEC or any other particular language task. Choosing to use pre-trained models has many good reasons. As discussed below in the section on transfer learning, exploiting the similarities between tasks in natural language saves resources and time.

We utilize the `t5`-family (Raffel et al., 2020) of pre-trained seq2seq models in our experiments. The models are presented in more detail in the following subsections. We choose this particular model family because it has a byte-level version of it that allows us to focus on the aspect we are interested in: language representation levels.

2.3.1 Transformers

Now we go into a bit more detail on the transformer architecture introduced in section 2.1. The transformers caused a revolution in NLP when they were first suggested in the paper "Attention is all you need" (Vaswani et al., 2017). Before this, most machine learning on sequenced data (like language in NLP) was done with sequential models like Recurrent Neural Networks (RNN) and Gated Recurrent Units (GRU).

The transformer is a sequence-to-sequence model at its base, which means that both the input and the output are sequences of some sort. It consists of an encoder and a decoder (see figure 2.1. They are trained with self-supervised and supervised learning. The crucial difference between RNNs and the transformer is that the transformer operates on a whole sequence at once, and not sequentially through the tokens. This is done with the help of the self-attention mechanism, which draws global relationships between source and target tokens, and between the tokens of each of

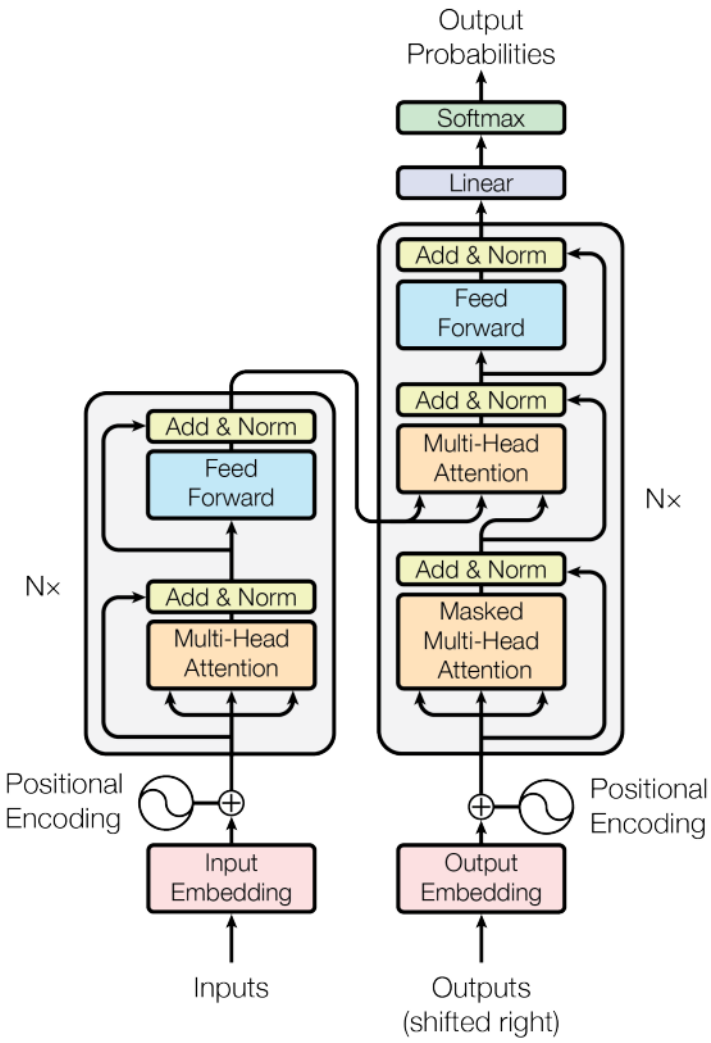


Figure 2.1: An illustration of the transformer architecture. Picture from Vaswani et al. (2017): "Attention is all you need". This is the architecture used by `t5`, which again is used in this thesis.

them separately. This means that it can do parallel operations to a higher degree than RNN-type models, and simply train much faster. In Vaswani et al. (2017) the researchers show that they can perform state-of-the-art translations from English to French at one fourth the cost of previous state of the art models.

A famous derivative of the transformer model is the Bidirectional Encoder Representation for Transformers (`BERT`; Devlin et al., 2019), which, as the name implies, only utilizes the encoder part of the transformer. Since `BERT` only uses the encoder, it does not output a text sequence. Rather it outputs a numerical representation of the input sequence where the individual tokens representations are affected by their context. These type of representations are called contextualized embeddings, and are a central part of modern NLP.

2.3.2 Transfer learning

Transfer learning exploits the similarities that are between different NLP tasks. The concept involves two main steps. First, a model goes through *pre-training* on one task. Later, the model is *fine-tuned* for another, target, task. This methodology has two advantages. First, fine-tuning an already pre-trained model can be less costly than a from-scratch training directly on the target task. Second, there could be limited training data for a particular target task, which would cause sub-optimal performance. A model that already contains some understanding of the language, can help increase performance on the target task.

2.3.3 t5

The pre-trained model `t5` (Raffel et al., 2020) gets its name from the 5 t's in "Text-to-Text Transfer Transformer". This model uses the transformer architecture which was outlined in section 2.3.1, and is pre-trained (transfer learning, section 2.3.2) on several NLP tasks. In addition it is trained on the enormous C4 corpus, which stands for Clean Common Crawl Corpus, in an unsupervised way. The important aspect about `t5` is that it converts all NLP problems into a text-to-text format, where the input is a sequence and the output is a sequence as well. This is opposed to models that are classifiers which choose from a set of classes as predictions. The exact way the model is trained is outlined in the subsections below.

Looking at the leaderboards on shared tasks on GEC we see that `t5` is one of the most popular starting points for GEC. It is also used in ensemble with other models with great success, for example in M. Qorib et al. (2022).

The architecture of the base `t5-base` model is as follows. The encoder and decoder parts of the model are of the same size, consisting of 12 blocks each. It uses SentencePiece (Kudo & Richardson, 2018) to encode text into a numerical representation on a subword basis. The other models have slightly different architectures, we refer to Xue et al. (2021) and Xue et al. (2022) for details on this.

Together with `mt5` (Xue et al., 2021) and `nort5` (Samuel et al., 2023), `t5` is the model we use as subword-level point of comparison with the byte-level `byt5` (Xue et al., 2022) model. `mt5` has the advantage of being the most similar to the byte-level model on many parameters. They are both trained on the same multilingual dataset. Since the level of representation is the core difference, we can easier determine how representation affects performance. `t5` on the other hand is mostly trained on English data, with a small percentage of Romanian, German and French text. It is also trained on some supervised tasks, which is not the case for neither `byt5` or `mt5`. But despite of these differences we still want to use `t5` as point of comparison for `byt5`, because it has shown good performance on GEC, for example in M. Qorib et al. (2022).

All the models in the `t5` family are released in different sizes. These sizes are determined by how many trainable parameters there are in their architecture. If nothing else is mentioned, we always use the `base` versions,

but there are smaller and larger versions available as well.

Unsupervised training of `t5`

`t5` is pre-trained on a self-supervised task called *denoising* or *masked language modelling*. This mirrors the way BERT (Devlin et al., 2019) is trained. During pre-training, 15% of tokens in a sequence are dropped or hidden at random, and constitutes a span. If multiple tokens come after each other, that becomes one span. These spans are substituted by a sentinel token. The model is then trained to predict what should be in those missing spans. This kind of training does not need any other targets or golden standards than the text itself. The dataset used for the unsupervised training is C4 (Raffel et al., 2020).

Supervised training of `t5`

This part of the training is sometimes referred to as fine-tuning. This is not to be confused with the fine-tuning we do in this thesis to develop the models performance on GEC. After the unsupervised training on huge amounts of data, the model is trained on a wide variety of NLP tasks. This includes question answering, text classification, machine translation, and summarization. This is the transfer learning part of the training process, as the "knowledge" it acquires during this part is assumed to transfer knowledge from one task to another.

One of the supervised tasks `t5` is trained on is that of Linguistic Acceptability assessment, with the CoLA (Warstadt et al., 2019) dataset. During work on this thesis we have created such a dataset also for Norwegian: NoCoLA (Jentoft & Samuel, 2023). You can read more about this project in chapter 3.

2.3.4 nort5

`nort5` (Samuel et al., 2023) is a very recent version of `t5` that is trained on Norwegian text, and pre-trained at the Language Technology Group (LTG) of the University of Oslo. `nort5` is trained with the same scheme as the English `t5`, with pre-training on self-supervised text-to-text masked language modeling. In the paper, the model is benchmarked on four generative tasks against `mt5` (the multilingual version of `t5`) and `north-t5` (`mt5` further fine-tuned on Norwegian text found online). The results are strongly in favor of the new model, with even the smallest version over-performing the larger versions of the old models by a large margin on three out of four tasks.

The model is openly available in four different sizes at `https://huggingface.co/ltg/nort5-base`. We use it together with `mt5` as a subword tokenized point of comparison for the Norwegian experiments with `byt5` in chapter 5.

2.3.5 mt5

mt5 (Xue et al., 2021) is one of the models we compare our byte-level model to. It is a multilingual version of t5, and just as t5 it uses subword-level language representation. It is pre-trained on over a hundred languages on the multilingual dataset named mC4: multilingual Colossal Cleaned Common Crawl. The training of the model mirrors that of t5 closely, but differs both in that is was trained on a multilingual corpus, and that it was only trained with the mask language modelling objective without any supervised training. Therefore, the model cannot be used in a zero-shot manner, i.e. it has to be fine-tuned before it can be used for any task that is not prediction of masked words.

mt5 was competitive in the multilingual benchmark XTREME (Hu et al., 2020) when the paper came out in 2021, but has been surpassed by several competitors since then. The main reason for using it in this thesis is that it gives us an opportunity to compare it one-to-one with the byt5 model described in the next section, as the pre-training schemes of these two models are very similar.

Just as t5, mt5 uses the software SentencePiece to tokenize text on a subword basis.

2.3.6 byt5

byt5 (Xue et al., 2022) is a version of mt5 that has been trained on the exact same data (mC4), but which uses a byte level representation of text for the input. Therefore, comparing byt5 and mt5 makes for an ideal opportunity to compare different representation levels. Some arguments for why byte-level models might be beneficial are presented in section 2.2. The model has a very small vocabulary for input (256), and handles all the complexity in the later layers of the model. Some adjustments have been made compared to t5 and t5. While the other two models have a balanced amount of parameters in the encoder and decoder, byt5 has more parameters in the encoder. byt5 does not conduct any text-processing prior to training, and it feeds the UTF-8 encoded text straight into the model as a byte representation. The model creators have also made some adjustments to the pre-training tasks, we refer to the paper for more details on this. Otherwise the models are very similar. But as one of the main goal of the researchers was to prove the usefulness of byte-level representation, the training scheme have been left as similar as possible to mt5.

The model has been shown to work well on noisy data. In Xue et al. (2022) the writers show that the model outperforms mt5 both "on word-level tasks sensitive to spelling and/or pronunciation, and in the presence of various types of noise". Therefore, we hypothesise that the model might work well for the GEC task, where we assume there to be a lot of noisy data. The noisiness in our task comes from words or phrases that have not been seen in the training data of the model, like wrongly spelled words, unlikely combinations of words and morphology that is inappropriate for a particular word.

2.4 GEC for Norwegian

As far as we know, no open research or models have been published on GEC for Norwegian using modern neural techniques. The SCARRIE project (Povlsen et al., 1999) worked with creating spell checking and grammar checking tools for all three Scandinavian languages in the mid-nineties. As of 2023, the project does not seem to have resulted in any publications or openly available products. The popular grammar checker for English, *Grammarly*, does not support Norwegian as of 2023. A special tool for children with language impairments: *Lingdys*[2] is offered to private customers and schools. We have tested a free demo version, and it only seems to correct misspelled words, and is not a GEC system as we have defined it in section 2.1. The company Tansa[3] offers a collection of tools to facilitate writing which includes what they call grammar control. They claim that they can detect *common* mistake types not only on word level. The tool is too expensive for us to use, and we do not find any open reviews of the product. We do not know what kind of technological architectures are behind their system.

As long as one has enough training data, the model architectures that are used in this thesis have base characteristics that make them more or less language independent. The experiments and results in chapters 5 and 6 show how relatively successful GEC systems can easily be developed for Norwegian.

[2]https://lingit.no/
[3]https://www.tansa.com/no/

Chapter 3

Datasets

3.1 General requirements

Figure 3.1: "Fresh (the) air is good for the body and the health" - example of sentence with error from the ASK parallel corpus

For training a system as described in chapter 2 we need a collection of pairs of uncorrected (source) and corrected (target) sentences: a parallel corpus. Uncorrected source sentences that possibly contain errors are not necessarily that hard to come by, since children, people with various language impairments, and language learners produce lots of possibly faulty texts all the time. But until we have automatic GEC systems that match human performance, the target corrections have to be created by humans, which is difficult and costly. Therefore such parallel datasets are not trivial to obtain, and GEC is sometimes described as a low resource task (Junczys-Dowmunt et al., 2018). Efforts have been made to collect and create such datasets, especially for English. Every day, foreign language students are generating texts that contain errors, and if these texts are corrected by professionals, a parallel dataset can be created. Especially the two last shared tasks on GEC have made some such datasets openly available.

In addition to presenting the datasets used in our experiments, this chapter discusses how the errors in GEC datasets can be annotated, how the lack of data can be compensated for with synthetic data generation, and how we are using our datasets for evaluation. Lastly, we show how the Norwegian ASK (Tenfjord et al., 2006) dataset can be used for evaluating

large language models, another big important topic in NLP.

A quick note on terminology. The different elements of a parallel corpus and the output of a model are described in various terms. For the original learner sentence, or a synthetic imitation of such, terms like original, source, unacceptable, input or learners text are used. The human annotated correct version can be called corrected, target, acceptable, output or reference. The latter can even be described as input in the context of feeding it into a neural network, as opposed to the output prediction of the model. The output of the model can be called output, prediction or hypothesis. Even if the terms source and target are a bit technical in nature, we try to stick with those for consistency throughout the thesis.

3.2 Error annotation

```
<p>Thank you very much for the prize <NS type="RT"><i>at</i><c>from</c></NS>
camp California in America.</p>
```

Figure 3.2: An example of a sentence in the xml-format, with one replacement error marked. Source: the CLC-FCE corpus

```
S This are a sentence .
A 1 2|||R:VERB:SVA|||is|||-REQUIRED-|||NONE|||0
A 3 3|||M:ADJ|||good|||-REQUIRED-|||NONE|||0
A 1 2|||R:VERB:SVA|||is|||-REQUIRED-|||NONE|||1
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||2
```

Figure 3.3: An example of a sentence in the M2-format, with multiple annotators (Bryant et al., 2019). The "noop" edit indicates no change by that annotator.

Knowing the types of errors in the source sentences is not a base requirement for a GEC dataset that is to be used for training neural network model. It is still useful in some cases. When developing GEC systems, one might want to know what types errors are present in the datasets, and how models behave when it comes to specific error types.

In manual error annotation a trained annotator marks error types into specific token spans in learner text. One common format for storing such annotations is eXtensive Markup Language (XML). See figure 3.2 for an example of this format. Error annotation can also be directly and automatically derived from the difference between a source and target sentence. The implicit M2 format is claimed to be the most common data representation style for grammatical error correction in Y. Wang et al. (2021). It accommodates corrections from multiple annotators. The

original sentence (S) is presented in its original form, and below it each line (A) represents one correction (span), marked as offset in the original sentence. Other included information is the error type and the correct string. See figure 3.3 for an example on the use of M2. The M2 format should not be confused with the MaxMatch GEC evaluation system, which is sometimes shortened to M^2.

M2 is automatically created from a source sentence combined with either a target sentence, or a system prediction. When one has both a source-target and source-prediction M2 file, these can again be compared to assess how well a model performs compared to the target. The scorer named ERRANT (Bryant et al., 2017), which is further described in chapter 4, does evaluation of GEC systems in exactly this way.

3.2.1 Annotation challenges

As mentioned earlier, parallel corpora for GEC require human annotation. Humans have to make the correct versions of the sentences, and this causes some challenges for the consistency of annotations. Annotating GEC is a challenging task for several reasons.

Firstly, one can strive to generate grammatical sentences with as few corrections as possible, or aim for the maximal fluency of the corrected sentences. These strategies are called *minimal* and *fluent* corrections respectively (Bryant et al., 2023). The ASK dataset, which we base our Norwegian GEC dataset on, is annotated with the minimal correction principle.

Secondly, it can be challenging to have full consistency on alignment of edit spans. In the sentence "*The apple red is*" there could be either one

$$red\ is \rightarrow is\ red$$

or two

$$red \rightarrow is$$

$$is \rightarrow red$$

edit spans. This variation can occur both within the annotations of only one annotator, or between annotators (Bryant et al., 2023).

Thirdly, one can annotate on sentence level or based on a bigger span. In the case of CLC-FCE and ASK, which are our primary corpora for English and Norwegian respectively, the annotators have corrected full learners essays, but most development of GEC systems from the datasets have been based on independent sentences. This causes noise for the models, as a correction might only be valid with knowledge on previous context. Sentence tokenization and change of sentence boundaries also causes misalignment.

Lastly, there is no agreed standard on how to categorize different types of errors. The schemes have varying numbers of subcategories, from less than thirty in NUCLE (Winder et al., 2017) to almost a hundred in CLC-FCE (Nicholls, 2003).

3.3 Data sparsity

One of the challenges in modern, neural NLP is the need for huge amounts of data to train the models. For tasks where datasets do not need annotation, one can use so-called self-supervised or unsupervised training. In those cases there has been great success for example in the creation of large language models, since the enormous amount of accessible digitized textual data can be used for training without a human annotator providing gold targets. These systems do however have tendencies to recreate human biases. For an overview of the different ways human biases can affect machine learning systems, we refer to Shah et al. (2020).

The situation is very different in GEC and the related task of (neural) machine translation (NMT). The methods in NMT have inspired most modern approaches to GEC. Both tasks require parallel corpora, and the problem of too little data becomes apparent. We can first look at the case of NMT. Optimal parallel corpora like official bilingual documents are comparatively rare and have a limited domain, and other corpora like movie subtitles are noisy. Multilingual Wikipedia can also be used, but as an article in two languages does not necessarily have sentence level correspondence, these kind of corpora can better be described as *multilingual corpora aligned on the document level*. For GEC structured datasets with original and corrected sentences are needed. These kinds of datasets do not have any other real-world use other than for research, so they have to be collected and annotated specifically for GEC development or similar purposes. Lacking appropriate data is therefore a bottleneck for these kinds of tasks.

One central focus in the fields of NMT and GEC has been to loosen up this bottleneck by creating additional training data synthetically. In the case of GEC, this means automatically creating text with errors to be used as the source part of a parallel corpus. Hereafter we list some techniques that have been used to do this.

The primary strategy is to create parallel corpora from clean monolingual text, by corrupting it in different ways. This can be called *data augmentation*, *synthetic data generation*, or *artificial error generation*. Then the corrupted version can be regarded as source, and the clean, original text is the target. Many of the participants of the 2019 BEA shared task (Bryant et al., 2019) focus on the use of data augmentation in the papers published related to the competition contributions. This is a confirmation that the data sparsity problem is a bottleneck for modern neural GEC, since these participants are "chasing the dragon" when it comes to achieving state-of-the-art results.

The following methods can be called *noise injection*, as in injecting noisy or erroneous elements into clean text. On word level, one can use pre-existing spell-checking tools to create wrongly spelled words (Grundkiewicz et al., 2019). This can be done by reversely converting, with some probability, correctly spelled words into some of the most common misspelled versions of the same word. The sets that each replacement word is taken from is called a confusion set. If one would replace words with random words from the vocabulary, one get very unlikely sentence pairs, that do not prepare the system for the challenges of a test set or real world applications.

In Omelianchuk et al. (2020) other kind of grammatical errors are automatically infused into the target sentences with a certain probability, which then become the source sentences. There are 4 errors. For "append error", a random word is added to a random position, for "verb error", a random verb is replaced by a different morphological form of the same verb, for "delete error", a spurious word from a dictionary of commonly deleted words is added to a random position, and finally for "replace error" the two first actions are performed in the same position.

In addition to noise injection, another method is *back-translation*. In Kiyono et al. (2019) the authors focus on some different strategies for data augmentation for GEC. They describe a method where a sequence-to-sequence model is trained on the available parallel corpora, but with the correct sentences as source and the corrupt sentences as target. This model can then create corrupted sentences from any available text, which in turn can be used as source sentences for training a GEC model.

Lastly, a more creative strategy is termed *round-trip translation*. Here one relies on the idea that automatic language-to-language translation tools yield errors. In Kementchedjhieva and Søgaard (2023), the researchers show that this way of augmenting data can improve the performance of medium sized models, but not on large models. For pre-training, they channel clean English sentences through another language and back to English again, and use the output as a source sentence and the input as target.

We have not explored any of the aforementioned techniques in our thesis. We have, however, expanded the amount of sentences available for training our Norwegian GEC models by extracting the ASK-EXPANDED (see section 3.4.3) version of the Norwegian ASK dataset, which doubles the amount of available training data. The obvious shortcoming of this strategy is that the "new" sentences have a lot in common with the "old" ones, consisting of the same vocabulary and the same errors. The experiments in chapter 5 show that this is not necessarily a fruitful strategy, especially not when evaluating on natural datasets as opposed to a dataset in the one-error-per-sentence style of ASK-EXPANDED.

In summary, data augmentation and creation of synthetic training data is a central part of the development of robust GEC systems today. Changing only individual words from correct to incorrect is not enough to cover the range of mistakes found in natural, faulty, text. If creating text mirroring actual faulty text, one has to add grammatical and semantic errors on sentence level as well.

3.4 The datasets we use

Table 3.1 gives an overview over the datasets that we use in this thesis. In addition to containing the required parallel sentences, the datasets differ in what metadata they have attached to the mistakes, and some don't have any at all (NAIST). Either the mistakes in the original texts are marked explicitly

(e.g. with an XML-system), or the mistakes are implicit in the sense that they can be detected by comparing the original and the corrected text with a tool such as ERRANT. There can also be some metadata about each text, such as the age of the writer, the country of origin (if the writer is not writing in their native tongue), an the proficiency level of the writer in the target language.

For the training of a system, we only need the source and the target. No other explicit features of the source are taken into account, as neural networks implicitly learn the most useful features in the early layers. But to be able to do an error-wise analysis of the models at a later stage in the research, annotations of mistakes can help discover patterns in how the system works.

Corpus	Sentences	Words	Errors	Err/sent	Corr/sent	Level
NAIST (Eng)	1 047 384	na	na	na	0.564	varied
FCE (Eng)	32 841	523 730	52 992	1.61	na	B2
ASK (Nor)	47 145	753 755	94 000	2.00	na	B1/B2

Table 3.1: Basic statistics on the datasets we use to train GEC-models. Not all data is available for all of the corpora. "Corrections" are the number of users who have added a suggested correction for the sentence as a whole. The values of the cells marked with "na" are either not obtainable, or need to be extracted with a error-alignment software, for example ERRANT.

3.4.1 CLC-FCE

Error-code	Explanation
S	spelling error
RV	replace verb
RP	replace punctuation
TV	wrong tense or verb
RT	replace preposition
MD	missing determiner
MP	missing punctuation
R	word or phrase needs replacing

Table 3.2: Explanations for the most common error types in the CLC-FCE corpus

This dataset presented in Yannakoudakis et al. (2011) consists of answers to prompts, written by candidates who have taken the Cambridge ESOL First Certificate in English (FCE) examination in 2000 and 2001. The publicly released subset of the dataset, named FCE-public, is the part we have access to, and it consists of 1,244 exam scripts. The dataset is created based

on a test to prove that learners are eligible for work or study in the UK, corresponding to the CEFR level B2. The dataset comes as scripts with two essays each. Each submission/script is in XML-format, with the mistake types explicitly marked. The essays are marked with a grade, which can be used to differentiate between essays with many or few linguistic problems. Some essays need many corrections, and some don't.

An error-code system with 88 different types of errors (Nicholls, 2003) is used. Each error is marked with one or several letters. The first letter usually describes the general problem: whether something is missing or lacking. The second letter describes word class. Other error codes do not follow this logic, for example "AGA" is anaphoric pronoun agreement error. The most common error codes are listed in table 3.2. For the rest of the codes, please refer to Nicholls (2003). We do not use these error codes for error analysis in chapter 5, because of the problem that arises when there are multiple errors in a single sentence. The scorer that we use for evaluation, ERRANT, cannot take anything else than source, target and prediction as input. Priorly annotated error tags from the dataset cannot be added. So aligning the error types with the predictions becomes impossible, and we will rather rely on the fine-grained analysis possibilities in ERRANT itself.

Converting CLC-FCE for GEC training

As is common in most GEC research, our models operate on sentence level. Each sentence is considered a completely independent training example. CLC-FCE comes in a essay separated structure in the XML format. Extracting the original sentences as source, and the corrected versions as target, from this is straight forward. This is, as long as the corrections are not related to sentence tokenization, where the corrector wants to split up or merge some of the source sentences. When this happens, we risk losing the alignment between source and target sentence. We have therefore relaxed the condition of only operating on sentence level, and we have always extracted the longest version, whether that is the source or the target sentence. In the rest of this thesis, the term *sentence* must therefore be understood more broadly as a sequence or phrase, possibly a bit longer than a single sentence.

For tokenization, i.e splitting the flowing text into tokens, we have used word_tokenizer from the Natural Language Tool Kit (NLTK) (Bird et al., 2009).

3.4.2 NAIST

The `NAIST` dataset, also called `LANG-8` dataset presented in Tajiri et al. (2012) is created from learners texts on lang-8.com, a free language-exchange social network. There are 1 million sentences, and it is currently the largest dataset suitable for GEC. The suggested target sentences are supplied by non-professional users. The metadata is: number of corrections added by users, original text (sentence), a unique id for each sentence and a unique id for each collection of sentences. Just below half (0.491) of the

sentences have corrections, and there can be several corrected versions of the original non-native sentence. Therefore the mistakes are implicit, and must be extracted with some automatic tool. The corrections might come from several contributors, as many of the corrections are identical. If a sentence has alternative corrections, we can expand them to separate training examples. The two main limitations with this dataset is that the mistakes are not categorized in any way, and that the corrections are made by non-professionals, which is a phenomenon often described as "noise" in the literature. The basic statistics about the corpus are in table 3.1. The ERRANT software, mentioned in section 2.1.1, can be used to extract the error types from just the source and target sentences.

NAIST is considered *the noisiest* of the publicly available GEC datasets in Bryant et al. (2023). To counter the noisiness, Rothe et al. (2021) have made a cleaner version of `NAIST LANG-8` which they call `cLANG-8`. The method was to use their best GEC model on the target sentences of the original corpus to exclude non-fluent language created by non-trained correctors. Unfortunately, we were not aware of this improved version in time to use it in our experiments in chapter 5.

We made several changes to NAIST in preparation for GEC training. We only made use of the *train* part of the dataset, and ignored the much smaller *test* part. We truncated the training part of the dataset from 1 000 000 to 40 000 sentences to have reasonable training times. For the sentences with multiple suggested targets, we picked only the first, since our evaluation scheme does not accommodate multiple hypotheses for comparison with the prediction. We did not perform tokenization on the data, as the originals where already tokenized.

3.4.3 ASK

We are thankful for the University of Bergen and Clarino for granting us access to the ASK corpus for this project. Without this corpus, we could not have developed GEC training corporas or GEC models for Norwegian.

This dataset, presented in Tenfjord et al. (2006), consists of about 50 000 sentences written by non-native language learners at two different levels of Norwegian knowledge. The dataset consists of essays corrected by professionals. The essays are written as solutions to two separate Norwegian language exams. The levels are estimated in Berggren (2019) to approximately CEFR-levels B1 and B2. For the essays, there is a rich collection of metadata, like age, native language, age, and occupation. Each mistake is also coded according to an error-coding system. A description of each error-tag can be found in figure 3.3. There is not complete agreement between annotators, as sometimes a separate xml-tag "del" and "add" is used to mark removal or addition of words, while other times the M and R error-markers are used with the most common "sic" tag for the same purpose. The use of the different strategies are merged in figure 3.4.

The ASK correctors are instructed to use the principle of minimal correction when correcting the learners sentences. This means that the goal is to achieve a grammatical sentence with as few corrections as possible,

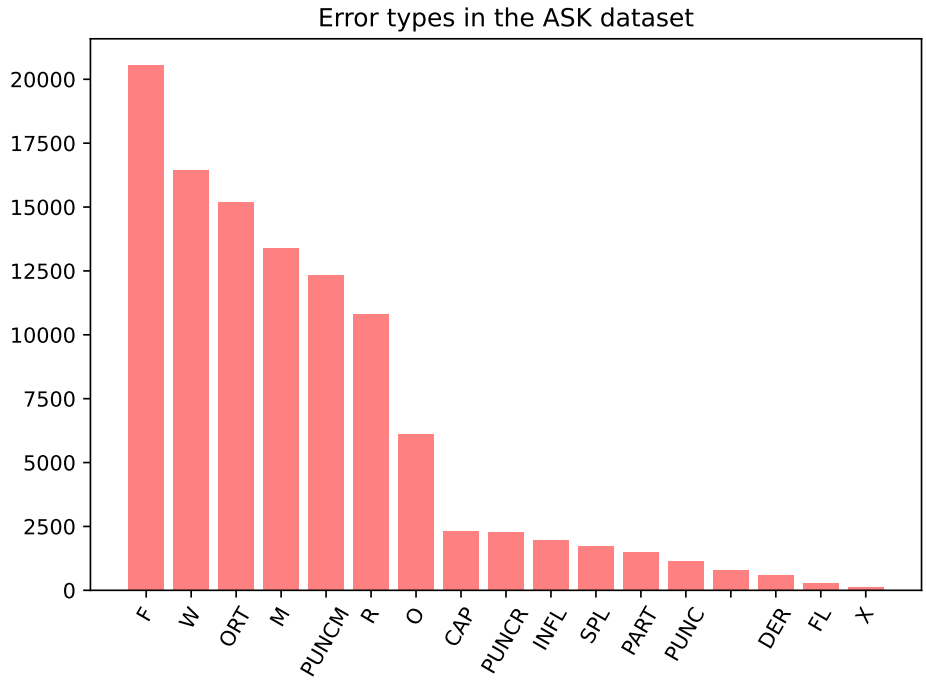


Figure 3.4: Distribution of error types in the ASK dataset. The bar with no label represents errors with no error tag marked.

and not necessarily maximising the fluency of the target sentence.

The dataset is also annotated with a CEFR-score (for parts of the set), which can be used as gold labels in the task of Automated Essay Scoring, as has been done in Berggren (2019) in their master thesis on automatic essay scoring. The dataset can also be used for NLI: Native Language Identification, because the data is annotated with the authors L1.

The distribution of error types in the dataset is shown in figure 3.4. Some basic statistics about the size is shown in table 3.1. These statistics are picked out before any train/test split has been done. There are approximately the same amount of errors per assignment in the B1 and B2 level submissions.

Preparing ASK for GEC training

When converting ASK for GEC training, we have taken some precautions to preserve the anonymity of the original authors. The corpus is annotated with rich metadata about the learners. We have decided to surpass all this metadata, including the CEFR-level of the test. ASK has already gone through a anonymization process, where possibly sensitive words have been replaced by placeholders. Still, some of the topics of the essays deal with so specific topics about the lives of the learners, that we decided to sentence-scramble the essays to achieve maximum anonymity.

For the GEC datasets we want sentences as the unit for evaluation. Therefore we need to split the continuous text of ASK into sentences. However, since some of the corrections suggested by the correctors affect

Error-code	Explanation	Additional comment
F	wrong morphosyntactic category: wrong form or inflection.	the code INFL also exists, which is used mostly for over-generalization
W	wrong choice of word	
ORT	orthography: spelling mistakes	
M	missing word: for pronouns and function words mostly	for content words a special tag <add> is sometimes used
PUNCM/ PUNC/ PUNCR	punctuation missing/ punctuation missing/ superfluous punctuation	If the missing element is a full stop, the corrected sentence consists of two original sentences
R	superfluous word	superfluous punctuation is a separate error type
O	problems with word order on word or phrase level	
CAP	error with letter casing (Det/det)	
PART	mistakes regarding compound words	SPL is for splitting up compounds, a common mistake for Norwegian learners
FL	word from other language than Norwegian	

Table 3.3: Explanations for some of the error types in the ASK corpus

the way the text is split into sentences, and we need alignment between the source and target sentences in the pairs for GEC training and evaluation, we decided to always keep the longest available version in cases where there is disagreement between source and target. Thus, the unit referred to as "sentence" in this paper can consist of multiple sentences, just as is the case for CLC-FCE.

We conducted a few additional post-processing measures to the dataset. All sentences are heuristically detokenized and removed if they contain an uneven count of quotation marks. If no error type is mentioned for a given correction, we also remove that sentence. The sensitive words that have been replaced by placeholders like "`@sted`" (place) and "`@navn`" (name) are replaced with a substitute representation of that category, i.e. "Oslo" instead of "`@sted`", to normalize all sentences. This is to avoid feeding too many tokens to the models which they have not seen during pre-training. In rare occasions, these replacements might cause some sentences to become erroneous, since the possible genitive and plural conjugations in the original texts are not annotated with separate placeholder-tokens.

ASK-RAW and ASK-EXPANDED

We have made two versions of the ASK GEC corpus: ASK-RAW and ASK-EXPANDED. ASK-RAW is a straight-forward extraction of source and target sentences from the corrected learner essays. In ASK-EXPANDED, for each sentence, we first extract a corrected version. Then we generate one source sentence for each error found in the original source. If one or zero errors, only one pair is generated. With this method we extract almost 100 000 sentence pairs, as many of the original sentences have multiple errors. This means that there are multiple sentences in the source category that only differ on one parameter, and multiple target sentences are the same.

We have made these variations on the ASK GEC corpus for two reasons, one related to training and the other to evaluation. Firstly, to compensate for the the data scarcity problem discussed in section 3.3, we wanted to have more training examples to study how that might affect performance. Secondly, we want to enable access to error types when evaluating sentences. To be able to align error types with prediction results, we require sentences to contain maximum one error each. Read more about how we use this dataset for evaluation in section 3.6.

3.5 Datasets in other languages

In our current uni-polar world, English language is by far the primary focus for model development in NLP. But efforts are being made to create parallel GEC datasets also for other languages. An Arabic GEC dataset is presented in Zaghouani et al. (2014). It consists of about 1600 documents. The annotations are made by trained annotators, and the source sentences are both from second language learners and native speakers. For Chinese language there are several resources and a shared task available, a more thorough overview can be found in Bryant et al. (2023). Czech, Japanese, Ukrainian, German and Russian languages also have parallel corpora available to them. The Norwegian ASK-derived corpus will hopefully be considered part of this collection in the future.

3.6 Datasets for evaluation

The purpose of this thesis is not to chase state-of-the-art performance on GEC in English. Rather, we want to study particular aspects of training and development. Therefore we have not used the official test sets from the shared tasks, which would have been the currently best way to assess the absolute predictive power of our models. Our results should not be directly used for comparison outside the scope of this thesis. Instead, for both English and Norwegian, we have created our own splits on the data, with 80% of the data being used for training, 10% is used as evaluation data during development and hyper-parameter fine-tuning, and 10% being withheld for the final test rounds.

For English, we only use CLC-FCE for evaluation. The predictions on development and test data are fed directly to the ERRANT tool (Bryant et al., 2017) for comparison with source and target sentences.

For Norwegian, we use two separate datasets for evaluation: the test subset of ASK-RAW and the test subset of ASK-EXPANDED. Both can be used for a estimation of performance on the relevant metrics with ERRANT. But in addition, the subset of ASK-EXPANDED can be used for error-type level analysis. Since each sentence has a maximum of one error, we can annotate each error with a single error type. Then we can estimate the models performance on those error types, based on whether the whole sentence was successfully transformed from source to target. From now on, we call this evaluation scheme ASKEVAL to distinguish it from ERRANT. This is the scheme that is used for fine-grained error analysis in chapter 6.

3.7 NoCoLA - another application for a parallel corpus

As has been discussed previously in this chapter, parallel corpora for machine learning are not trivial to come by, as opposed to monolingual free text, which is abundant on the internet. Now that we have actually found a source for such parallel data in Norwegian with erroneous sentences and their corrected counterparts, we can think of other uses for this data in NLP.

3.7.1 Language model evaluation

Modern large language models like BERT (Devlin et al., 2019), GPT-3 (Winata et al., 2021) and ELMO (Peters et al., 2018) have become familiar to the wider public in recent years, especially after the user interface ChatGPT became available in 2022. These models are used directly, or further fine-tuned, for a variety of tasks. Therefore there is a need for a strategy to quantitatively evaluate their overall quality and capabilities. In English the General Language Understanding Evaluation (GLUE) benchmarking suite (A. Wang et al., 2018) is commonly used for this purpose. It includes evaluations on nine tasks, including sentiment classification and question answering. Among the datasets used is the Corpus of Linguistic Acceptability (CoLA) (Warstadt et al., 2019), which is used to test the models ability to distinguish acceptable sentences in English from unacceptable ones. The corpus consists of 10 000 sentences that are annotated as either acceptable or unacceptable. The large language models are fine-tuned to classify these sentences into one of the two categories.

3.7.2 NoCoLA

In recent years large language models have been developed even for Norwegian, like `NorBERT1`, `NorBERT2` (Kutuzov et al., 2021) and `NB-BERT` (Kummervold et al., 2021). During the work with this thesis we decided to contribute to the development of Norwegian language models by creating

Dataset	Train	Dev	Test
NoCoLA$_{class}$	116 195	14 289	14 383
NoCoLA$_{zero}$	—	—	99 115

Table 3.4: Number of sentences and sentence pairs, respectively, for both the NoCoLA datasets.

the benchmark NoCoLA (Jentoft & Samuel, 2023), the Norwegian Corpus of Linguistic Acceptability. It is based off the ASK-derived datasets we created for grammatical error correction, with some adjustments. We are happy to note that NoCoLA has recently been included in the benchmarking suite NorBench (Samuel et al., 2023) created by the Language Technology Group at UiO.

There are two versions of the dataset, and the sentences counts for each are shown in table 3.4. For NoCoLA$_{class}$ all the ungrammatical sentences from the learner essays are annotated as unacceptable, and both the grammatical learner sentences and the corrected versions of the ungrammatical sentences are annotated as acceptable. For evaluating a language model, it is fine-tuned on a binary classification task. The standard fine-tuning approach from Devlin et al. (2019) is used for this.

For NoCoLA$_{zero}$ all the learners sentences with more than one error are modified to contain only one error each. This results in multiple unacceptable sentences for each original sentence with more than one error. These sentences, and the ones containing only one error, are mapped to their corresponding acceptable sentence. This was done so that one can identify the type of error that distinguishes the two sentences in each pair, and thereby get a more fine-grained description of a language models performance. Learner sentences without errors are discarded. Now we have a collection of pairs of unacceptable sentences and their acceptable counterparts. Since language models natively give probabilities for language sequences, language models can be evaluated by how often they give a higher probability to the acceptable sentence in a zero-shot manner.

The modification done to the multi-error sentences mirrors the way we create the EXPANDED version of the GEC dataset, described in section 3.4.3.

3.7.3 Comparing NoCoLA to GEC parallel datasets

There are many similarities between the NoCoLa datasets and the GEC datasets, for example that both applications demand a source for unacceptable sentences. The difference between NoCoLA$_{class}$ and our GEC-dataset is that in NoCoLA$_{class}$ the unacceptable sentences are not coupled with their acceptable counterparts. The sentences are simply lumped into one of the two categories and shuffled. When a model under evaluation sees a sentence, it has to judge whether it is unacceptable without seeing the other version of it.

The difference between NoCoLA$_{zero}$, which is also a parallel dataset, and

our GEC-dataset is that NoCoLA$_{zero}$ does not contain the sentences from ASK that did not have any mistakes. This is because one cannot make a judgment of acceptability between two identical sentences, which would be the case if the target is the same as the source in such cases. In GEC on the other hand we want the models to be able to preserve the source sentence if it is correct, so we need to have some negative examples. Otherwise the models in production would be inclined to change every single input sentence, even if it was grammatically correct and acceptable. Another difference to the raw version of our GEC dataset is that any unacceptable sentence contains only one error.

Chapter 4

Evaluation

4.1 Evaluating seq2seq

To be able to create good automated NLP systems we need to have a clear goal of what we want to achieve, and how to measure whether we have achieved it. This requires us to have some sort of golden standard, or target, that is manually created. We want to be able to do the measurement automatically, not by having humans inspect and judge every single output of the models. For tasks where the possible outputs is a closed set, this is relatively easy. One example would be when developing a model to classify documents into a closed set of non-hierarchical and mutually exclusive classes. If the model outputs the correct class for unseen documents, it succeeds in its task. It is either yes or no, a binary evaluation. When aggregating multiple predictions, one can count successful and unsuccessful predictions, which again give rise to such metrics as accuracy, precision, recall and f1-scores. These metrics give us an understanding of the models performance, and become proxies for how good it is.

But for sequence-to-sequence (seq2seq) types of tasks, having a closed set of classes that the prediction belongs to is usually not as optimal. Human language is generative and practically unlimited in its creativity, as claimed by linguists like Noam Chomsky (1965). If human language is unlimited, then how can we limit the possible successful predictions of a model to a set of outputs or a specific reference sequence? This is the first challenge of evaluating seq2seq tasks like Machine Translation (MT), summarization, and GEC. For translation tasks there can be multiple good translations that convey the semantic meaning of the source language sentence well, but which differ in lexicality, morphology and syntax between themselves. The same goes for summarization, where a longer text or paragraph can be summarized in multiple ways. For grammatical error correction, different corrector instructions can lead to vastly different target possibilities. Creating only one specific target sequence can make the evaluation too strict if doing a binary decision, because it does not allow for this variation.

Researchers have worked on coming up with solutions to alleviate these problems. One obvious way is to create multiple reference sequences. But the workload for annotators increases linearly for each new reference, and it

is difficult to decide the right time to stop creating more reference sentences. In neural machine translation metrics that compare substrings between target and prediction are dominant, specifically the BLEU score (Papineni et al., 2002). In addition to span-comparison it has a brevity penalty, to avoid too short predictions. In NMT the source and target are usually from completely different languages, and any overlap between target and prediction is rewarded. For summarization, according to Fabbri et al. (2021), a metric derived from BLEU called ROUGE is the default automatic evaluation metric. It combines overlap between target and prediction with a measure of the fluency of the prediction. As we will see in the following section, some other considerations have to be taken when evaluating GEC.

4.2 Evaluating GEC

The difference between machine translation and GEC is that, depending on the proficiency level of the original text, in GEC there are many similarities and overlappings between source and target. Sometimes even whole sentences might be identical, as the source text does not have any errors. Using a pure span-based strategy like BLEU would reward a system too much for those similarities, to the extent that even a copying machine would get a high score. Most decent systems would be scoring at close to 1.0, without much room to differentiate between them. The scores of a given system would also vary widely depending on the language proficiency of the source sentences.

Therefore, the most common, evaluation for GEC is correction-span based. This means that the similarity between source and target sentences are ignored, and only the spans where prediction and target differ from the source are compared and used for evaluation. This also compensates for differences in proficiency levels of source text, as it is the relative portion of *corrections* that are made, and not of total number of words, that is counted.

4.2.1 Level of evaluation

In both the recent shared tasks GEC has been framed as a sentence level task. This means that during training and evaluation, each sentence is treated as an independent unit without taking context into consideration. But as the corrections have usually been made based on a larger context, a misalignment arises. A corrector might correct *He loves ice cream* into *She loves ice cream*, if the preceding sentence in the source text is *This is Annabelle*. This would introduce noise into the parallel, sentence-wise, dataset. Usually it means that the evaluation becomes too strict if the system has not made a correction in the above example, even if the context available to it indicates that the prediction is in fact grammatical.

To deal with this problem, some researchers have explored to perform GEC on document level (Chollampatt et al., 2019). Their models strive to take a wider context into consideration by implementing an auxiliary encoder for modeling previous sentences. Sentence level training and

evaluation is still the standard in GEC research, and it is how the task is defined. Some of the issues that still cause the task to be defined on sentence level are that our current models cannot deal with sequences of much longer length than sentences, and that we lack of corpora the appropriate structure (Bryant et al., 2023). But this might change in the future. For example in Náplava et al. (2022) a GEC dataset for Czech is released in three alignment levels: sentence, paragraph and document, in anticipation of possible future work with larger contexts.

4.3 Metrics

Figure 4.1: Simple example on how the source, target an prediction are used to calculate error-span based scores

4.3.1 $F_{0.5}$

One possible source of confusion needs to discussed. Even if error-span based $F_{0.5}$ can be called *a metric* regardless of the methodology for how to obtain it, the survey literature, for example Bryant et al. (2023) and Y. Wang et al. (2021), describe the different ways to obtain it as *metrics*. In this thesis we reserve the term *metric* only for such concepts as accuracy, precision, recall, $F_{0.5}$ and I-measure, while we use *scorer* for the methodologies to obtain them. The scorers are discussed in section 4.4.

As mentioned earlier, correction-span based $F_{0.5}$ is the standard metric to evaluate GEC, and is the one used for both the benchmark shared tasks mentioned in section 2.1.3. For each span of words that make up a correction, the source, target and prediction are compared, see figure 4.1. True positives (tp) are corrections matching the target, false positives (fp) are corrections made by the system that should not be there, and false negatives (fn) are corrections that the system missed. Notice here that true negatives (tn) are indirectly ignored, which means that correctly uncorrected sentences do not affect the score other than by subtracting examples from the other three possible outcomes. From aggregating the counts we can get precision: $^{tp}/_{tp+fp}$ and recall: $^{tp}/_{tp+fn}$. The formula for deriving $F_{0.5}$ from this is:

$$f0.5 = \frac{(1 + 0.5^2) * precision * recall}{(0.5^2 * precision) + recall}$$

.

For GEC, the most common is to value precision higher than recall, so we go for $F_{0.5}$ instead of $F_{1,0}$, which would value precision and recall equally. Weighting precision more than recall in this way is an indicator that in this task one would rather have some mistakes left uncorrected, which would decrease recall, than have many corrections that are not correct, which would decrease precision. Accuracy, another measure derived from the comparisons of prediction and target that also takes false positives into account, is useless in such as case. This is both because one can assume that in GEC the majority of text is error-free, and in any case, the rate between erroneous and correct text is unknown, so-called unbalanced classes.

There are however, several issues with using the $F_{0.5}$ score as well. When looking at the participant contributions of the ConLL shared task (Ng et al., 2014), the $F_{0.5}$ scores for the systems do not correlate with rankings based on human judgement. As human judgment is inherently hard to quantify, we can look at some of the features of the $F_{0.5}$ score that could possibly cause the low correlation. One of the aspects not addressed by $F_{0.5}$ are that ungrammatical edits are rewarded, although not as much as a grammatical edit. Another is that non-edits are not rewarded in any way, even if they can be argued to be an important part of the task. A third is that different kinds of edits are rewarded and penalized equally, while human judges might weight for example adding a comma as a greater error than adding an extra letter (Napoles et al., 2015).

4.3.2 Other reference based metrics

$F_{0.5}$ on error-spans seems to be the preferred metric for evaluating GEC. But there are other metrics as well. The improvement measure, or I-measure for short, is based on the alignment between source, target and prediction. It is a weighted accuracy score where True Positives and False Positives are weighted higher than True Negatives and False Negatives. It does not require explicitly annotated error types (Y. Wang et al., 2021). The Generalized Language Evaluation Understanding metric (GLEU) (Napoles et al., 2015) was derived from the BLEU score from machine translation. It was motivated by the fact that human-annotated edit spans are somewhat arbitrary and time-consuming to collect. It does not require explicit edit annotations but rather only corrected reference sentences. In GLEU prediction n-grams that match the target but not the source are rewarded, while prediction n-grams that overlap with the source, but not with the target, are penalised (Bryant et al., 2023). Compared to $F_{0.5}$ it thereby also rewards non-corrected words and sentences.

4.3.3 Reference-less metrics

Some strategies for evaluating GEC strive to avoid the use of manually defined references. This has the obvious advantage of bypassing the need for human annotations. Another motivation is that metrics based on target always penalize edits that are *not* included in reference, which can result

in underestimation of GEC systems (Y. Wang et al., 2021). Since we only use reference based evaluation in this thesis, we will now only look at one example of a reference-less evaluation scheme. Choshen and Abend (2018) proposes a measure of semantic faithfulness to evaluate the transformation from source to prediction. Their system compares the semantic symbolic structure of the source and prediction. This is done with help of a automatic semantic graph system called UCCA (Bryant et al., 2023). If the semantic structures in source and prediction are similar, a correction can be considered faithful. This system would rate a copying of source highly by its own, and is thus just meant as a complimentary evaluation scheme to other schemes.

4.4 Scorers

To obtain $F_{0.5}$ scores we need a automatic scorer. There are some practical challenges connected to calculating $F_{0.5}$. Detecting what constitutes a correction-span is not trivial. In the minimal case one needs to align the predicted sentences with the target sentences, decide what the unit of "correction" is, and then find the overlap between them. The alignment algorithm should ideally be linguistically motivated, which makes such a system language specific. One might also want a more fine-grained analysis, where different correction types are categorized directly from source, target and prediction. In its simplest form, this could be just add, remove, and replace.

4.4.1 MaxMatch

The MaxMatch (M^2) scorer (Dahlmeier & Ng, 2012) is the official scoring strategy used for evaluating the participants of the shared task from 2014 (Ng et al., 2014). It accommodates for multiple target sentences as options. It delivers a span based $F_{0.5}$, and emphasises the need to have maximal overlap between the edits of the hypothesis and the edits inn the multiple reference sentences. This method relies on Levenshtein alignment (Levenshtein, 1966) to explore and find this maximal alignment.

4.4.2 ERRANT

ERRANT (Bryant et al., 2017) is a fine grained, rule-based, correction-categorization scorer developed for English GEC. It is a two-stage automatic standardization process. It has 25, mainly part-of-speech based, error categories. In addition all of these can again be subdivided into replacement, omission and insertion, giving a total of up to 75 different error types. It was developed by Bryant et al. for the shared task in GEC in 2019, and is thoroughly described in Bryant et al. (2017). It is officially only created for English, and has language specific tools for POS-tagging, stemming, and alignment. It uses the python package Spacy (Honnibal & Montani, 2017) for this purpose. This is the package we use to evaluate the English part of our GEC experiments.

For evaluation of our models we have used ERRANTs command line interface. When conducting inference, the predictions are written to text-files, and ERRANT produces two files in the M2 format (see chapter 3): one for the changes from source to target and one for the changes from source to prediction. These two M2 files are then compared, and the metrics are calculated. We utilize ERRANTs option for granularity to access separate scores for different error-categories in chapter 5. Some examples of the ERRANT output are to be found in the appendix of this thesis.

4.4.3 ERRANT for Norwegian

```
Example 1:

S Dette er det miljøet som liker jeg.
A 5 6|||R:NOUN|||jeg|||REQUIRED|||-NONE-|||0
A 6 7|||R:NOUN|||liker.|||REQUIRED|||-NONE-|||0

Example 2:

S Der må man besøke noen adresser og kan man kjøpe nesten hva som helst.
A 5 6|||R:SPELL|||adresser,|||REQUIRED|||-NONE-|||0
A 7 9|||R:WO|||man kan|||REQUIRED|||-NONE-|||0

Example 3:

S Den er veldig fint skrevet, og jeg leste boken selv.
A 7 7|||M:NOUN|||har|||REQUIRED|||-NONE-|||0
A 7 8|||R:SPELL|||lest|||REQUIRED|||-NONE-|||0
```

Figure 4.2: Examples of ERRANTs treatment of Norwegian. This format is called M2, and shows the original sentence with a list of changes needed to be made to transform the source sentence to either reference or hypothesis sentence.

In this thesis we are using the ERRANT scorer for Norwegian *as is*, meaning we are not making any adjustments to the software. The software is not able to categorize error types at all, like in example 2 in figure 4.2 where a punctuation error is categorized as a spelling error. This it because it relies on linguistic cues and word lists. The overall scores seem to be reliable. Looking at the result tables in chapter 5, the results for Norwegian correspond reliably with the ones for English. The scores should, however, not be read as absolute scores for GEC performance, both because the test set is not a officially defined one, and because ERRANT shows some errors in span-detection for Norwegian. In errors with word order, it sometimes defines two errors where there is only one, like in example 1 in figure 4.2. For English, such a switch of word order would have been considered one error. But at other times ERRANT correctly defines a two-word span, and only registers one error, like in example 2. In example 3 ERRANT registers 2 errors, while the more correct analysis would be that there is one error of the *replacement* type. The tense of the verb should be changed, and the past perfect just happens to be two words in Norwegian, as opposed to

the preterite one-word. It is hard to automatically estimate the rate of such errors.

Therefore the performance scores between languages are not comparable, also because we use different evaluation datasets. ERRANT is still useful to compare the performance of different models. The fine-grained analysis offered by ERRANT is ignored for Norwegian.

4.4.4 ERRANT for other languages

A version of ERRANT for the Czech language is published in Náplava et al. (2022). They have adapted the original error types, and added a few language specific error types. A German version is available at Boyd (2018). Belkebir and Habash (2021) presents ARETA, a version of ERRANT for Arabic. They aim to address morphological richness and orthographic ambiguity of the language. Those peculiarities cause challenges, especially when Arabic with and without diacritics are mixed. There is also a version of ERRANT for Greek.

None of the adaptations of ERRANT for other languages than English are official. We have decided that doing the necessary adjustments for creating a Norwegian ERRANT is too time consuming, and have left that for future work. See chapter 8 for a discussion on what this task might require.

4.4.5 ASKEVAL

Since we do not have an ERRANT for Norwegian, we have developed a scheme for the purpose of fine-grained error analysis in chapter 6 which we call ASKEVAL. With the help of the one-error-per-sentence ASK-EXPANDED evaluation dataset (see chapter 3 section 3.4.3), we can align error types with prediction success. We only count a successful prediction (TP) when it is identical to the target sentence. If the source and prediction are identical, but not the target, we count a False Negative. If the source and target are the same, but the prediction is different, we count a False Positive. Here we diverge from the principle of only comparing error-spans. But this can be justified when we know that the systems are good at copying the source, and that there is only one error in focus per sentence. The obvious caveat is that the predictions might have errors in other parts of the sentence than the focus. That will lead to an unfair amount of False Positives.

4.5 Conclusion

Evaluation on GEC is still a field of active research. The same is the case for evaluation of other seq2seq tasks as well. In this chapter we have discussed the pros and cons of different metrics and ways to obtain them, as well as the challenges connected to using the existing tools for Norwegian GEC.

As none of the scorer-delivered metrics for English have been conclusively shown to correlate optimally with human judgment, Bryant et al. (2023) recommends using different scorers and metrics for different test

datasets. This is to have a fair comparison between systems, as the scores, even the $F_{0.5}$ scores of different scorers, do not correlate perfectly with each other. As the goal of this thesis isn't to challenge the state of the art systems for GEC, and we do not evaluate on one of the benchmarking test sets, we choose to use the ERRANT scorer because it allows us to study metrics on specific error types.

Chapter 5

Experiments and Results

In this chapter we go through the experiments we have done with fine-tuning seq2seq models from the `t5`-family for grammatical error correction. The prime objective is to compare the byte-level `byt5` model to the subword-based `t5`, `mt5` and `nort5` models. First we go through some general considerations regarding the way we conduct the fine-tuning and evaluation of the models. Then we look at the performance of the models when fine-tuning mainly on the English FCE-CLC dataset, and then on the Norwegian ASK dataset. During the English experiments we try out different flavours of pre-trained models, changing training hyperparameters, use different sizes of datasets, and test out ways to facilitate training large models. In the Norwegian experiments we simply run the same pre-trained models on our Norwegian ASK dataset, changing the English `t5` to the Norwegian `nort5` model. Finally we summarize the results achieved during the chapter.

Since we have experimented with quite many combinations of models, datasets and hyperparameters, we have numbered the rounds of training from 0-7. If the reader wants to see the results of our final models directly, these are reported in round 5 for English and round 7 for Norwegian.

5.1 Methodology and general considerations

5.1.1 Pretrained models

For performing our experiments we fine-tune language models to do the GEC task with the help of appropriate dataset of the parallel type, described more thoroughly in chapter 3. We have chosen to use pretrained models in the `t5`-family of sequence-to-sequence (seq2seq) models. The models have been pre-trained on unsupervised and supervised tasks. They have been used extensively for fine-tuning for GEC before, and many of the recent state-of-the-art models are based on `t5`. You can read more about the architectures of these models in chapter 2. One such example of the successful use of `t5` on GEC is in Rothe et al. (2021). Their best model achieved an $F_{0.5}$ score of 68.87 and 75.88 on the ConLL14 and BEA19 shared tasks respectively.

5.1.2 Evaluation data

If nothing else is mentioned, all evaluation results are obtained on a withheld development set. Later in rounds 5 and 7 we are using test datasets. All evaluation data is simply a subset of the parallel datasets that is not used by the systems during training, and does not affect the models in any way.

5.1.3 Metrics and reported numbers

For all experiments we report results with three different metrics. All of them are based on comparison of source, target and prediction. The important aspect of GEC evaluation is that we only compare corrections, and ignore the tokens in sequences that are identical between source, target and predictions. The first is precision, which is how many of the corrections made by the model are the same as in the reference. The second is recall, which tells us how many of the corrections expected by the reference that the model managed to capture and correct. The third is $F_{0.5}$, which is a function of precision and recall that weighs precision higher than recall. You can read more about these metrics and the justifications for them in chapter 4.

For some of the runs we also report training time, which is relevant when making decisions on what models to use. If the performance difference is minuscule, but the training time (and resource) difference is huge, one might want to consider using a model setup that is slightly weaker in performance. If other hyperparameters are varying between the experiments within a round, those are also reported in the tables.

5.1.4 Software and hardware

For training neural networks, using Graphical Processing Units are shortening training time significantly, on the order of 100x. The reason for this is that the job such units are specialised on, namely accelerating graphical rendering for video games, is quite similar to the linear algebra operations used for training neural networks. Students at the University of Oslo have access to the high-performance computing platform Educloud-FOX. This is a platform shared by students and researchers, and opens up for running multiple experiments at the same time on high speed.

Software-wise we have used the framework PyTorch (Paszke et al., 2019). It is a framework for building deep learning models, written in Python. It allows us to abstract away from the actual matrices of parameters, and focus on the relevant parts of the work.

For evaluation we use both the ERRANT scorer (Bryant et al., 2017) for English *and* Norwegian as well as a self-made evaluation scheme, ASKEVAL, for fine-grained analysis of error type-wise performance only for Norwegian.

5.1.5 Model hyperparameters

Some considerations and decicions need to be taken when fine-tuning a pre-trained model like `t5`.

Number of epochs

One epoch of training is one run through the whole training dataset. If a model is trained for one epoch, all source and target sentences are only fed to the model once. But as the weight updates are quite small, it can be useful to go through the dataset more times, and for all our experiments we have used either 5, 10 or 15 epochs. The specific number is mentioned together with the results in each round.

Regularization

All the `t5` models we have used have a default dropout rate of 0.1. This means that during training, 10% of elements in the hidden vectors are skipped or *dropped out* temporarily and at random. This has been shown to decrease the chance for overfitting the model to the training data, as no single parameter gets too much to say to decide the prediction. Rather, the predictive power is distributed among the parameters. Dropout was first introduced to neural networks under that name in Hinton et al. (2012), and further popularized in Srivastava et al. (2014).

Learning rate

For each forward-pass in the neural network, a gradient is calculated, which tells us the *direction* each parameter should go to improve the performance of the model. The learning rate is the value the gradient value is multiplied with when updating the parameters of the model. A high learning rate makes big changes on the model at the time, while a small learning rate does small changes.

With models in the transformer-family, it has been showed that starting the training with a high learning rate is detrimental to the result. See Popel and Bojar (2018) for some empirical evidence for this. The solution is to use an optimizer with *warmup*, meaning that the learning rate starts low, and reaches its highest value after about 10% of training time, and then decreases. An optimizer is a part of the program that updates the weights of the neural network. We have used this methodology throughout all experiments, as this has been shown to be successful in other transformer contexts, also the in the original transformer paper (Vaswani et al., 2017).

Generation hyperparameters

During training with a transformer type model, the predictions are not generated autoregressively, but in parallel. When generating the predictions for inference the tokens are generated autoregressively, and some hyper-parameters can be chosen. We are using the default parameters which come with the models. Some of the most important ones are listed in table 5.1.

Parameter	Value	Description
max_length	50 tokens when subword, 250 bytes when byte-level models	Maximum length of generated sequence
min_length	10 for both	Minimum length of generated sequence
do_sample	False	Each next token/byte is chosen with greedy search

Table 5.1: These are some of the generation hyper-parameters that come as default with the models in the `t5` family. We have not modified them when generating with our models.

5.2 English

As a source of training sentences we use the CLC-FCE dataset (Yannakoudakis et al., 2011), with a 80/10/10 split. This means that there are about 40 000 sentence pairs in the training data, and the rest of the data is for evaluation purposes. We are using the native `t5`-tokenizer to convert the input sentences into integers in the `t5`-vocabulary, as well as adding padding tokens for the batched input.

5.2.1 Round 0 - `t5` without fine-tuning

Model	Precision	Recall	$F_{0.5}$ score
`t5`-base	0.043	0.043	0.043

Table 5.2: These are the results for `t5`-base without any fine-tuning, i.e. zero-shot. Remember that the $F_{0.5}$ score metric is based on correction spans and, input-copying ability is ignored.

Input	Target	Prediction
This is a first class hotel.	This is a first-class hotel.	The staff were very helpful and friendly.
I am questioning myselfs - am I sick man?	I am asking myself - am I a sick man?	- am I sick man?? - am I sick man?
It happened at the end of school year.	It happened at the end of the school year.	It happened at the end of school year.

Table 5.3: Some examples of the predictions when using `t5`-base without fine-tuning. The target column indicates the desired outcome

As the `t5`-model (Raffel et al., 2020) is a text-to-text model by nature and from pre-training, it already outputs something based on input by default. As a baseline and comparison point we want to test how well

t5-base performs without any fine-tuning, also called zero-shot prediction, on the Grammatical Error Correction task. The hyper-parameters for the prediction generation are outlined in section 5.1.5.

For some particular tasks one can use a prefix (concatinated with the input string) to direct the model in a specific direction. The task are summarization (having a multi-sentence text as input and outputting a shorter summary of it), translation from English to German, translation from English to French and translation from English to Romanian. These are the tasks that the model has been trained with supervised training to solve. Unfortunately GEC is not one of these tasks, so we do not utilize this feature, and feed the input sentences to the model as they are, without a prefix.

As we can see from the example-outputs in table 5.3 the input is relevant for the predicted output, and sometimes the prediction is a copy of the input. At other times, the model is creating output that is not a valid sentence. As our evaluation metric is quite strict and does not premiere just copying the output, the results as shown in table 5.2 are very poor.

We did not use mt5 for this baseline. The researchers from Google who released the model write on their Huggingface repository that mt5 was only pre-trained on mC4 excluding any supervised training. Therefore, this model has to be fine-tuned before it is usable on a downstream task, unlike the original t5 model [1].

So to conclude we can see that to be able to perform grammatical error correction, t5 needs to be fine-tuned for this particular task with a suitable parallel dataset.

5.2.2 Round 1 - basic t5

Model	Casefolding	Precision	Recall	$F_{0.5}$ score
t5-base	no	0.573	0.358	0.512
t5-base	yes	0.569	0.36	0.51
t5-large	no	0.581	0.382	0.526
t5-large	yes	0.579	0.384	0.526

Table 5.4: These are the results from round 1 of experiments. Model-sizes and casefolding is in focus.

The zero-shot prediction with t5 did not yield any good results. In this phase of experiments we try out how the base-sized pre-trained t5-model performs on our data after some epochs of fine-tuning. We also try a version of the model with more parameters, called t5-large. The disadvantage of this model is that the training and evaluation takes more time, and it is more demanding on the GPU of the training computers.

In addition we experimented converting the whole dataset to only lower case characters, hereby called *casefolding*. The motivation for this is a feature

[1] huggingface.co/docs/transformers/v4.16.2/en/model_doc/mt5

of this particular dataset, where a significant portion, 390 out of 19880 sentence pairs in the training data, are only in upper case. As the tokenizer of the `t5`-models is case-sensitive, this means that *CAT* and *cat* are two independent tokens from the models point of view. Our hypothesis is that the variation between the two forms prohibit optimal generalization in the training phase. One obvious argument against doing casefolding is that the model will not be able to distinguish between proper and common nouns, and that the model in "production" predicts sequences of characters all in lower case, which would be an undesired feature (bug) of the model.

Model	**Sentence**
	Example 1 - Articles - success
Source	Accident like that can really make you livid.
Target	An accident like that can really make you livid.
`t5`	An accident like that can really make you livid.
	Example 2 - Conjugation - success
Source	We aren't allowed to smoke and drink any alcohol.
Target	We aren't allowed to smoke or drink any alcohol.
`t5`	We aren't allowed to smoke or drink any alcohol.
	Example 3 - Verb-tense - failure
Source	I would be glad if you consider my suggestions.
Target	I would be glad if you considered my suggestions.
`t5`	I would be glad if you would consider my suggestions.

Table 5.5: Some examples of fine-tuned `t5` predictions in round 1 of experiments.

The results on prediction from this round can be found in table 5.4. We have added some examples from the predictions from the models trained in this round, these can be seen in table 5.5. The errors are categorised according to the authors intuitions, and do not necessarily coincide with any other error-annotation scheme. The main purpose of including the examples here is to remind the reader of the task we are performing, so as to not get completely lost in metrics and hyperparameters.

As we can see from the results in table 5.4, casefolding the sentences does not increase the performance of the models. With the disadvantages of casefolding in mind, we do not use casefolding in the rest of the experiments.

5.2.3 Round 2 - gradient accumulation

Now that we have seen the significant, but still minor, improvement achieved by using the large model instead of the base model, we return to using the base model. This is to save the shared training resources we have available.

As some of the models used have many millions of parameters which

t5-model checkpoint	Parameter count
t5-small	60 million
t5-base	220 million
t5-large	770 million
t5-3b	3 billion
t5-11b	11 billion

Table 5.6: A comparison of the parameter counts of t5-base versus t5-large

Model	Gradient Accumulation	Batch size	Precision	Recall	$F_{0.5}$ score
t5-base	no	32	0.564	0.364	0.508
t5-base	yes	8	0.562	0.363	0.506

Table 5.7: Studying the effects of using gradient accumulation. Each result is an average of 3 runs to exclude bad luck as a source of difference. For the second row the batch gradients are accumulated 4 times so that gradients are accumulated after 32 examples

need to be stored in the memory of our hardware, we experience crashes while fine-tuning some of the models. The more detailed reason for the problem is that as many training examples are processed at the same time in a batch, each example increases the workload linearly. The memory load on our hardware can be decreased by decreasing the batch size, but this can cause problems for the training. When a batch is small, the sample is less representative of the dataset as a whole. Thereby the direction of the gradients could be very different from the direction that improves the performance on the whole dataset.

For all experiments from now on we have implemented a trick called gradient accumulation (Kozodoi, 2021). Here we divide each batch into mini-batches,[2] and accumulate the gradients for each of the mini-batches before updating the weights of the network.

We compare a run with batch size 32, and one with mini-batch size 8 with 4 iterations before the updating of weights. The latter also adds up to 32 examples. As wee can see in table 5.7 the results are more or less equivalent, which means we can safely use this trick to decrease the batch size when needed. However, gradient accumulation makes the training slightly slower, but less than 10%.

The reason for the slight difference for the model without gradient accumulation from the experiments in round 1, is that we limited the number of epochs to 10, and truncated all sentences of above 150 sub-word tokens.

Sentences	Duration (h:m)	Data source	Precision	Recall	$F_{0.5}$ score
6 666 (small)	1:42	clc-fce	0.564	0.309	0.484
20 000 (standard)	1:58	clc-fce	0.564	0.363	0.508
60 000 (large)	2:59	clc-fce + naist	0.576	0.317	0.495

Table 5.8: These are the results from round 3 of experiments: studying the effects of training data size. All experiments are with the pre-trained model `t5-base`. *Duration* is the time it took to fine-tune the model, including intermediate evaluation steps.

5.2.4 Round 3 - training data modifications

Since Gradient Accumulation seems to be a valid way to modify the training procedure, we use it for the rest of the experiments.

In this section we want to study the effect of the size and quality of our training corpus. We create a small and a large version of the training data. The small, one third of the standard size, version consists of a subset of FCE-CLC. For the large version, three times the standard size, we have added a subset of the NAIST dataset, which you can read more about in chapter 3. Evaluation data is the same for all sizes of training data, and consists of the same development split from CLC-FCE used for evaluating all our English models.

From the results in table 5.8 we can see that using the standard sized training dataset is positive, while enlarging the data from NAIST is detrimental to the results. The resources saved by cutting the data down to small are quite insignificant compared to the loss of almost 2 points on performance.

The reason for difference in performance between standard and large need some explanation. The general trend in machine learning and NLP is that "more is more", which would result in a steady increase of performance when the training data size increases. In the worst case, the results should plateau at some point because of the variation in possible reference sentences. In our case the reason for the *decrease* in performance could be that there is a problem with the NAIST dataset, maybe related to the fact that the target sentences are crowd-sourced. This means that the target could be sub-optimal and noisy, which results in the model learning wrong corrections, or ignoring some mistakes that should have been corrected. Another possibility is simply that NAIST and CLC-FCE are two different datasets, with different prompts, learners with different backgrounds, and different subjects being written about.

Because of the poor results, from now on we surpass this crowd-sourced dataset completely. We also stick to the standard-sized dataset, as we do not think the time saved using the smaller subset to be worth the loss of performance.

[2]The terms batch and "mini-batch" are sometimes used to denote "the whole dataset" and "batch" respectively, but not so in this thesis

5.2.5 Round 4 - `mt5` and `byt5`

Model	Epochs	dur. (h:m)	LR	Precision	Recall	$F_{0.5}$
`t5`-base	10	1:58	default	0.564	0.363	0.508
`mt5`-base	10	1:50	default	0.471	0.141	0.321
`mt5`-base	10	1:49	0.00003	0.082	0.046	0.071
`byt5`-base	10	5:36	default	0.598	0.341	**0.520**
`byt5`-base	10	5:27	0.00008	0.562	0.33	0.493
`byt5`-base	10	5:34	0.00003	0.607	0.17	0.401

Table 5.9: Results after introducing `byt5` and `mt5`. Learning rate (LR) "default" is 0.0003. This is the learning rate that was used for all experiments so far in this chapter. All experiments with Gradient accumulation 8x4, except for `byt5` which uses 4x8.

Now we get to introduce the byte-level model `byt5` to fine-tuning for GEC. To make `byt5-base` (as opposed to `byt5-small`) work we needed to make some adjustments to the training setup. This is because the GPU cannot handle `byt5`-base de jure batch size of 8, so we switch it to 4 with gradient accumulation 8 times. As we have proven in a previous section, adjusting the gradient accumulation parameters does not significantly affect the results, so we can safely make comparisons between training-procedures on different de jure batch sizes when the de facto number of examples used for each optimizer update are the same

The multilingual model `mt5` is also introduced in this round, as a subword-level direct point of comparison to `byt5`. Theoretically there should be no advantage of using it over `t5`, as long as we assume that most of the training and evaluation data is monolingually English, which is to be expected in a dataset collected from an English learners test of super-intermediate level.

In this round of experiments we want to also experiment with some other learning rates, to see if this might increase the performance of models.

There is a significant increase in training, evaluation and prediction time when moving to byte-level systems. This is due to several reasons, but all relate to each sequence consisting of about 5 times more tokens. In the prediction and evaluation phases a calculation is made for the next token for each token, and thereby the duration is increased five-fold.

In table 5.9 all the results from this round are listed. With the learning rates we have chosen, the multilingual model `mt5` struggles to give good results. With the default learning rate, the one we have used for all experiments so far, `byt5` performs better than `t5`, while the other learning rates come short.

5.2.6 Round 5 - testing on test data

Now we have come to the focus of our experiments. These are the results we use to conclude on how byte-level models perform GEC compared to

Model	Epochs	Precision	Recall	$F_{0.5}$ score
t5-base	20	0.569	0.364	0.510
mt5-base	20	0.545	0.231	0.428
byt5-base	20	0.572	0.343	0.504

Table 5.10: Results when testing the three models on withheld test data. Scores are averages of 3 identical runs for each model type

subword-level models. We do two types of comparisons. The first is to compare byt5 to the model that is most similar to it on all other parameters except for the representation level: mt5. The second is to compare it to the subword-level model that is trained specifically on English data, namely t5. The latter is the more likely subword-level alternative for developing GEC, as the results are in the same range as those for the byte-level model.

In this round we turn to the completely unseen part of the datasets: the "test" data. In the preparatory stages of training there was a slight instability for byt5. Because of this, we run 3 fine-tuning runs for each model, with exactly the same hyperparameters, for 20 epochs each.

Lets first compare byt5 to mt5. From table 5.10 we see that the byte-level representation is clearly advantageous for overall $F_{0.5}$ score. Even if precision is emphasized in $F_{0.5}$, it is primarily the extremely low recall score of the subword-level model that pulls the overall score down. Recall tells us how many of the total errors the model managed to notice and correct. Since the precision is not as low, this means that mt5 tends to copy the input, but is too conservative when it comes to changing anything.

Then we can turn to comparing byt5 to t5. Some of the clear tendencies we can see from the results in table 5.10 are that the precision is pretty equal between the two models, but the recall of byt5 is a bit weaker. This means that the model copies the input well, the changes it does are correct, but it does not notice as many mistakes. Even if the recall is weighted lower than precision in $F_{0.5}$, the final result is slightly better for t5. In figure 5.1 we can see how the models learned during the epochs. t5 takes only one epoch to achieve stable and decent $F_{0.5}$ score, while the byte-level model takes more time. The high precision score early on for byt5 indicates that it is copying input, but not yet correcting many of the errors. Between epochs three to ten the model stabilizes, and precision falls together with the increase in recall.

In chapter 6 we look in more detail at the error-wise predictive capabilities of these models. For now we can conclude that byt5 and t5 are relatively equal on overall performance, while mt5 is falling behind and is not a competetive option for GEC in English.

5.3 Norwegian

We now turn to Norwegian language and fine-tuning the models on the ASK corpus. Using the experience gained during the English experiments, we go straight to comparing how the different models perform compared to

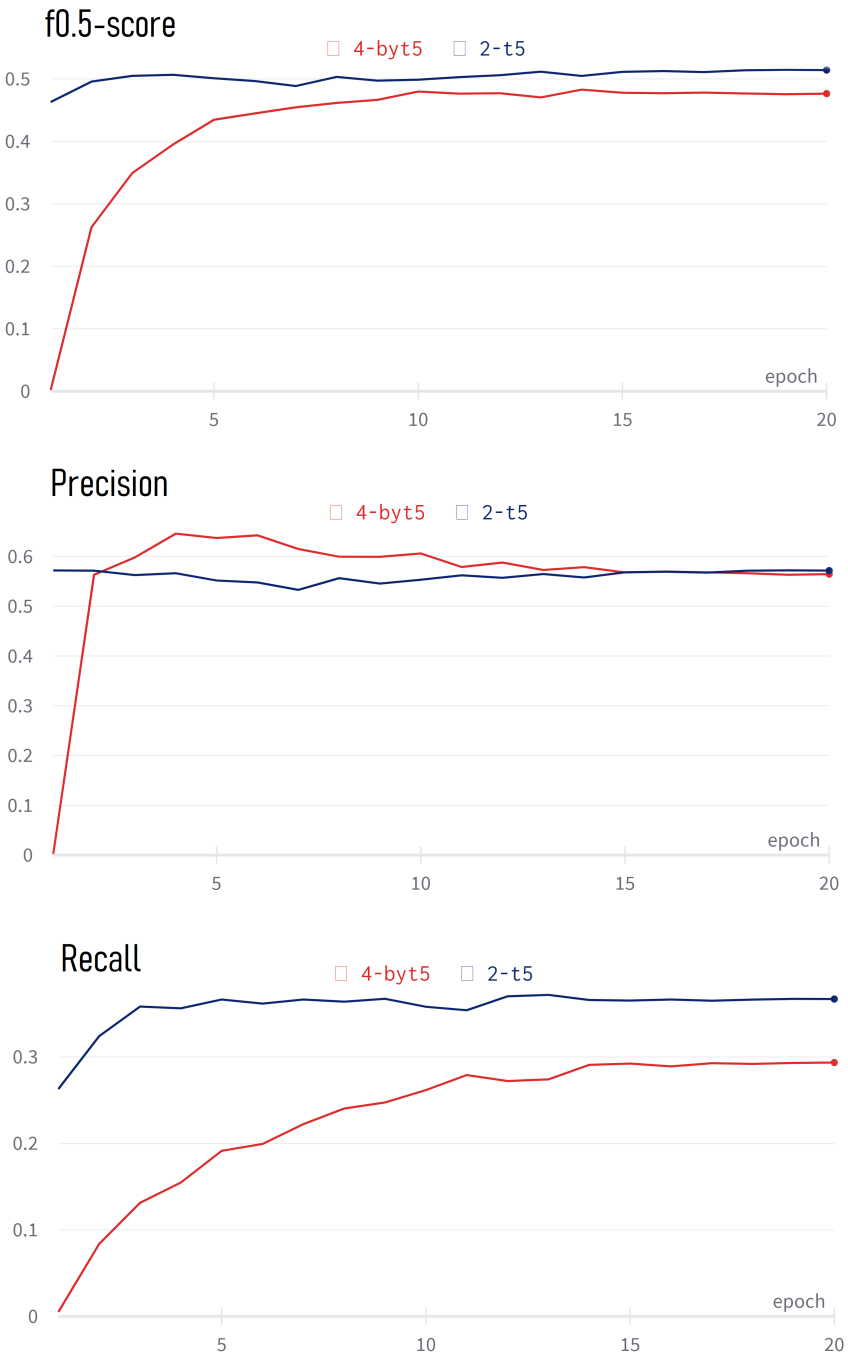


Figure 5.1: Round 5 (English): `t5` versus `byt5` on test data. Red color is `byt5`, blue is `t5`. The plot shows how the models scored when measured after each training epoch.

each other. We substitute the English-specific `t5` model with the brand new `nort5`, which is pre-trained on Norwegian data.

5.3.1 Evaluation for Norwegian

We use the ERRANT scorer for obtaining our metrics for Norwegian as well. As the tool is not developed for Norwegian language, some caution should be taken when looking at the evaluation metrics. You can read more about this in chapter 4 section 4.4.3. The fine-grained analysis offered by ERRANT is ignored for Norwegian.

5.3.2 On the datasets

First a quick note on the datasets themselves. In chapter 3 we described how we have created two versions of our ASK-derived GEC datasets, for both training and evaluation purposes. In the first version we use the original sentences, with their corresponding corrected versions. This is called *raw* or ASK-RAW. The second is a bit more complicated. Since each original sentence might have more than one error, we create a set of sentences which include only one of the error each. This is dataset is called *expanded* or ASK-EXPANDED. The sizes of the two datasets can be observed in table 5.11, repeating the table here for convenience.

Corpus	Total sents	Train sents	Dev sents	Test sents
ASK raw	45807	36534	4519	4757
ASK expanded	110947	88887	11168	10895

Table 5.11: There are two versions of the ASK dataset for GEC training. Here are the sizes of the two datasets, i.e. the sentence counts. We always evaluate on both datasets, while which training sets are used is varied.

When it comes to evaluation, we always use two separate evaluation sets, one from each of the aforementioned categories. ASK-RAW is the focus of our overall performance judgement, and ASK-EXPANDED is the one we use for error-type analysis in chapter 6.

5.3.3 Round 6 - Norwegian ASK - development

It is the *raw* evaluation we should focus on from table 5.12. This is from the testing on naturalistic data. We included the *exp* results because those are the ones we use when we dive deeper into the different error types in the next chapter. By including them here we get an idea of the correlation between the two evaluation schemes. You can read more about the different training sets in chapter 3.

Looking at the results in table 5.12, `byt5` is by far the model that performs the best on Norwegian GEC, regardless of the training data or evaluation scheme. The model only pre-trained on English data is much weaker than the rest. `mt5`, the most direct point of comparison to `byt5`, is significantly weaker than its byte-level counterpart. `nort5`, the subword-level model pre-trained solely on Norwegian data, performs surprisingly badly, and we get back to that subject in the next subsection.

Model	Training data	Precision raw/exp	Recall raw/exp	$F_{0.5}$ raw/exp
t5 (eng)	both	0.402/0.232	0.277/0.321	0.369/0.246
nort5	both	0.485/0.341	0.369/0.438	0.456/0.357
nort5	raw	0.532/0.373	0.417/0.467	0.504/0.388
nort5	exp	0.49 /0.4	0.268/0.444	0.42 /0.408
mt5	both	0.583/0.442	0.391/0.46	0.531/0.445
mt5	raw	0.57 /0.411	0.293/0.346	0.479/0.396
mt5	exp	0.588/0.468	0.34 /0.441	0.513/0.462
byt5	both	0.614/0.496	**0.465**/0.534	0.577/0.503
byt5	raw	0.631/0.482	0.451/0.507	**0.584**/0.487
byt5	exp	**0.636**/**0.565**	0.35/**0.554**	0.547/**0.563**

Table 5.12: Norwegian: trained on the ASK corpus. Evaluated on both raw-style sentences and expanded-style sentences. All models are "base"-sized. 10 epochs each for t5 and mt5, 5 epochs only for byt5. Remember that the results with ERRANT have some issues, read more in chapter 4.

The models (especially byt5) trained only on the expanded dataset (with maximum one error in each sentence) seem to be conservative in error-detection, and thereby have a high precision but lower recall. This makes sense because the training data is sparser on errors than the raw dataset.

The fine-tuning of byt5-models on our huge Norwegian dataset is very costly, and we have decided to only run them for 5 epochs instead of 10. The reason we feel confident to stop the training early for the byt5-models, is that the improvement stagnates already after 3 epochs. For the run with byt5-both the evaluation between epochs shows the following $F_{0.5}$-scores for the last three epochs: 0.574, 0.576 and 0.577. This pattern is representative for all three runs.

English t5 is included as a sort of baseline: the model architecture is appropriate for our seq2seq task, but the model is not trained on Norwegian data at all, and the tokenizer is English-specific.

Are there benefits from using any particular dataset or combination of datasets for training? How about if one takes training time into consideration? At a first glance it looks like training on ASK-EXPANDED or both is not advantageous. In chapter 6 we study these effects in more detail.

Improving nort5 with learning rate modification

Model	Learning rate	Precision raw/exp	Recall raw/exp	$F_{0.5}$ raw/exp
nort5	0.0001	0.598/0.459	0.465/0.523	0.566/0.47
nort5	0.00005	0.608/0.481	0.484/0.548	**0.579**/0.493

Table 5.13: Results when fine-tuning the hyper-parameter "learning rate" for nort5, because the initial experiments were worse than expected. Both runs use only the *raw* training dataset

The surprisingly low results on nort5 inspired us to experiment with some

of the hyperparameters to see if we can improve the result. The results were surprising, because the corresponding model for English performed at the same level as `byt5`. We run `nort5` with learning rate medium (0.0001) and low (0.00005). Since training only on the raw dataset gave us the best results on the raw evaluation (which we consider the most important), and the second best on the expanded evaluation, we only use the raw dataset for fine-tuning these two models with new learning rates.

The results when using these new learning rates can be seen in table 5.13, and show how a decreased learning rate is very beneficial for `nort5`. The moderately lowered learning rate improved the performance on all three metrics, and the significantly lowered learning rate even more so. Therefore we will keep this lower learning rate for the final evaluations on the test dataset in the next section.

5.3.4 Round 7 - Norwegian ASK - test

Model	Learning rate	Precision raw/exp	Recall raw/exp	$F_{0.5}$ raw/exp
`nort5`	0.00005	0.608/0.515	0.477/0.539	0.577/0.52
`mt5`	0.0003	0.568/0.426	0.295/0.341	0.479/0.406
`byt5`	0.0003	0.626/0.524	0.452/0.514	**0.581**/0.522

Table 5.14: Results from round 7 of experiments. Comparing `nort5`, `mt5` and `byt5` on the withheld ASK test dataset

We now move over to using the test splits of the ASK dataset. Here we compare our best versions of the `nor-t5`, `mt5` and `byt5` models. The predictions from these runs will be analysed in more detail in chapter 6. Also, the model checkpoints are released on Huggingface Hub [3] so that they can be used for inference by the reader under the names `norgec_nort5.pt`, `norgec_mt5.pt` and `norgec_byt5.pt` respectively.

Optimization of learning rate was only done on `nort5`. The other two models did not benefit from lowering the learning rate in our English experiments, so we stick to our default value for this hyper-parameter.

The final results are shown in table 5.14. Even with only 5 epochs of training, `byt5` performs the best on both testing corpora. It is its high precision score that is responsible for this, even if the recall is weaker than that of `nort5`. This result means that if one needs a language model only to perform GEC, it is not necessary to train a Norwegian-specific model to achieve good results. The multilingual `mt5` is weaker than the other two models on all parameters, but especially its ability to detect error seems to be weak, reflected in the low recall score.

In chapter 6 we will look at the error-wise predictive capabilities of these three particular models.

[3] https://huggingface.co/MatiasJ

5.4 Summary

In this chapter we showed how we have fine-tuned our GEC models, and what results different pre-trained models, hyperparameters and datasets have generated. When comparing `t5` and `byt5` on English GEC, we saw fairly similar performances when measuring $F_{0.5}$ scores, albeit differing in the distribution between precision and recall. This means that byte level representation in language models is a viable alternative when training GEC models. `mt5`, the subword-level representation version of multilingual `t5`, did not perform comparably to its byte-level counterpart.

We have presented the first GEC models for Norwegian, consisting of pre-trained models from the `t5` family fine-tuned on the Norwegian ASK parallel corpus. We have seen that `mt5` is not a viable solution to conduct GEC in Norwegian, but `byt5`, the version of `mt5` that has a byte-level tokenization, performs very well on GEC after fine-tuning. This shows that the best way to reap the benefits of multiligually trained models is to have byte-level representation, at least for Norwegian. And if the only requirement of a language model is to perform grammatical error correction, one can safely use a multilingual, byte-level model instead of training a Norwegian specific model from scratch.

In chapter 6 you can read more about the detailed performance the models had on different error-types. There we will see how the byte-level models handle different error types compared to subword-level models. We will also study in more detail how our experiments with dataset expansion affected performance.

Chapter 6

Analysis

In this chapter we aim to explore some of the research questions posed in chapter 1. In chapter 5 we saw that the overall performance of `byt5` matched `t5` and was significantly better than `mt5` on grammatical error correction. But how does representation level affect performance on an error-type level? And what did we learn from using the expanded training data when training our Norwegian GEC models?

All the analysis in this chapter is done based on predictions by the GEC models trained in chapter 5. In section 6.1 we use the ERRANT scorer to look at error-type-wise scores. In section 6.1 we use ASKEVAL to do the same thing for the Norwegian models. Finally in section 6.3 we discuss how our data augmentation tricks affected performance on our different evaluation sets.

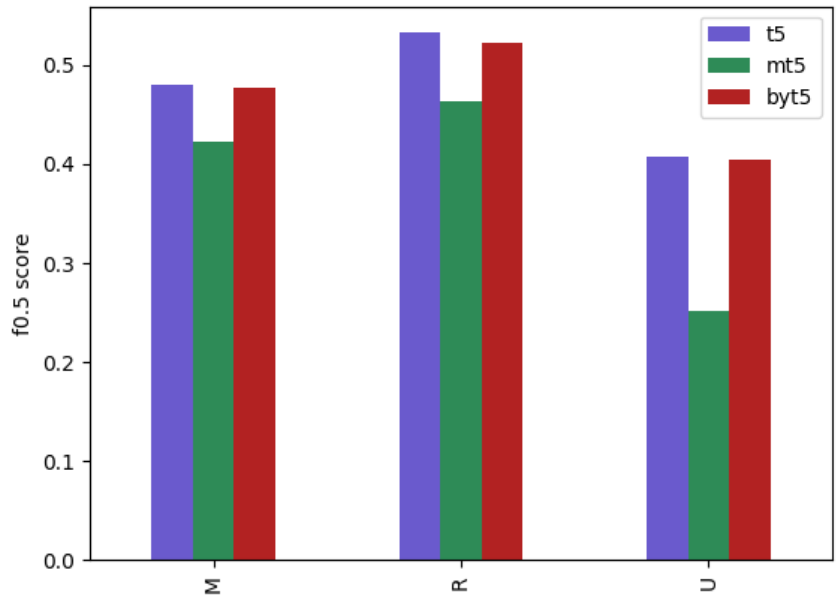
A note on terminology: if a model is described as for example `t5`, it is not the pre-trained model itself, but the fine-tuned GEC version.

6.1 English - fine-grained-ERRANT

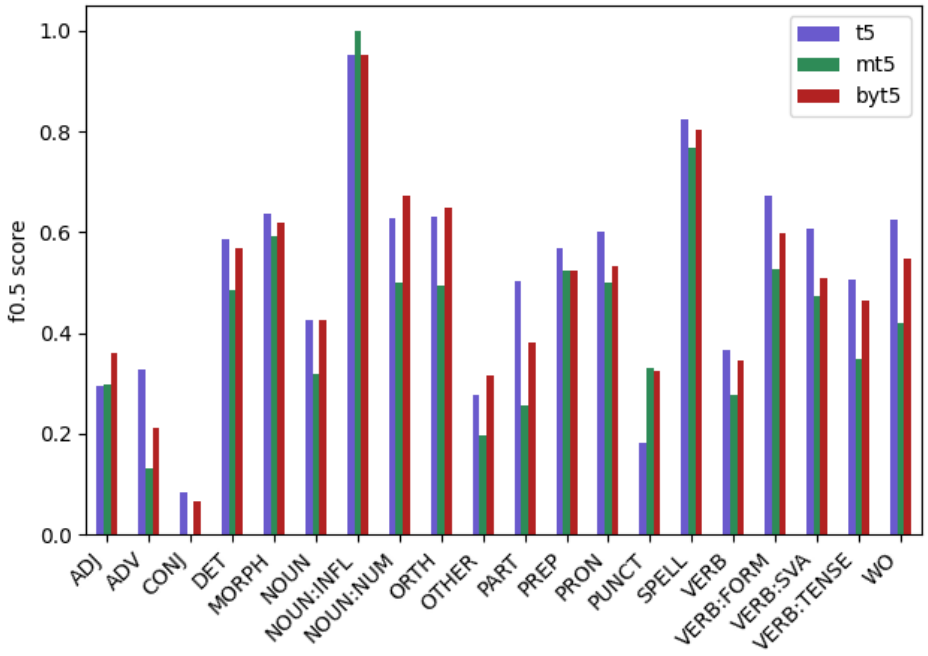
Abbreviation	Explanation	Abbreviation	Explanation
M	Missing	ORTH	Orthography
R	Replacement	PREP	Preposition
U	Unnecessary	PRON	Pronoun
CONJ	Conjugation	WO	Word order
CONTR	Contraction	SVA	Subject-Verb agreement
POSS	Possessive	PART	Particle

Table 6.1: Table explaining some of the possibly ambiguous abbreviations in figures 6.1 and 6.2

One hypothesis regarding byte or character level models is that they are good at dealing with noisy data. In the case of GEC, this could mean dealing well with error types such as wrongly spelled words and morphological issues, and the correction of those. It would be beneficial to see if this is

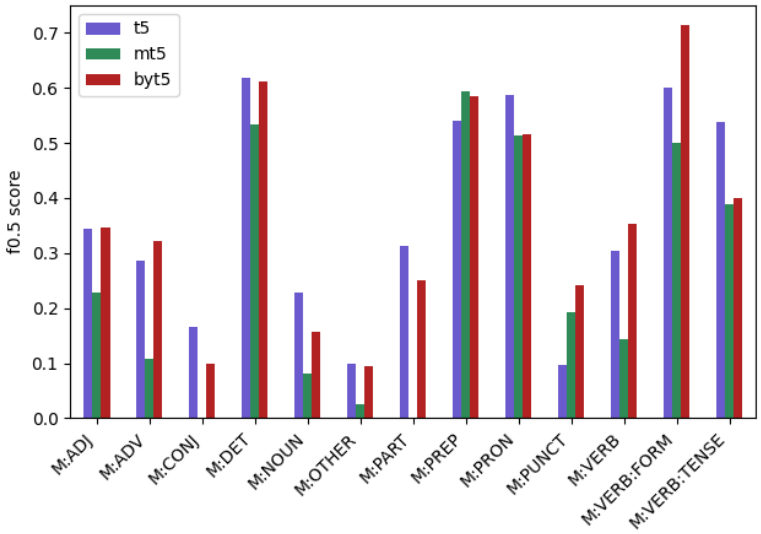


(a) `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: highest granularity level. Round 5: English

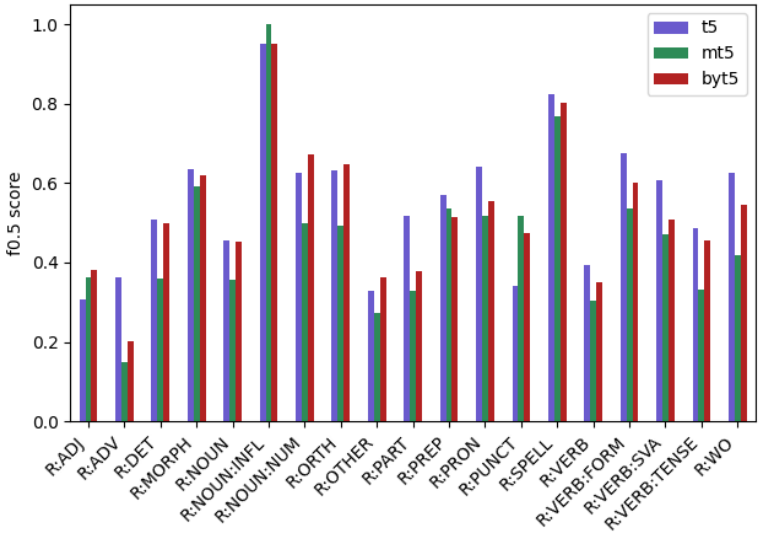


(b) `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: medium granularity level. Round 5: English

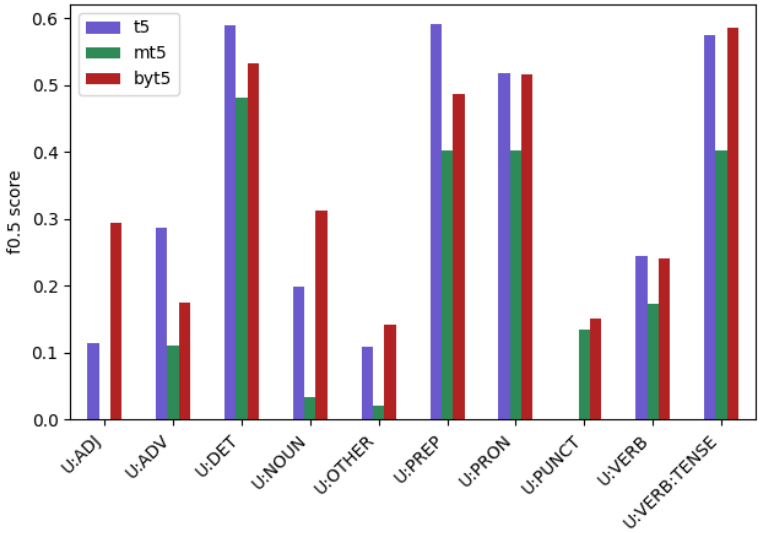
Figure 6.1: `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: high and medium granularity levels. Round 5: English



(a) `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: lowest granularity level: "Missing". Round 5: English



(b) `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: lowest granularity level: "Replacements". Round 5: English



(c) `t5`, `mt5` and `byt5` $F_{0.5}$ results on GEC by error type: lowest granularity level: "Unnecessary". Round 5: English

Figure 6.2: Errant English: lowest granularity level. The charts are split according to edit type: M/R/U

evident as a difference in performance between t5 and mt5 as compared to byt5 on some of the relevant subcategories of errors.

Now we will take advantage of the ability of the ERRANT (Bryant et al., 2017) scorer to look at the corrections in a more fine-grained manner. For this analysis we pick only one version of each of the three models, more precisely the one with the median $F_{0.5}$ score out of the three in round 5 of experiments on test data. For the overall results on that round see table 5.10. ERRANT gives us 3 options for how granular we want the evaluation to be. Level 1 (see figure 6.1a) only shows operation tier scores; e.g. R. (replace). Level 2 (see figure 6.1b) shows main tier scores; e.g. NOUN, but not operation tier scores. Level 3 shows all category scores; e.g. R:NOUN.

Please remember that as the overall ability of t5 is slightly better than byt5, we can expect the same slight difference for each score without concluding that t5 is better at this particular sub-category. The same goes for mt5 compared to the two other models. Its performance is overall significantly worse than the two others.

The full evaluation outputted by ERRANT on all fine-grained parameters for all three models can be found in the appendix, and in the bar-charts we only show a comparison of performance measured in $F_{0.5}$. In the appendix you can find counts for True Positives, False Positives and False Negatives, as well as scores for precision, recall and finally $F_{0.5}$. But it might be easier to just look at the $F_{0.5}$ scores compared in the bar charts in figures 6.1 and 6.2.

Let us first look at the patterns that are common to the three models. There is after all a great deal of correlation between them. All three models are best at correcting replacement errors, and worst at removing unnecessary words or phrases. mt5 has a noticeably lower score on unnecessary edits, with $F_{0.5}$ 0.251. The strongest second level categories for all models are errors in noun inflection, spelling, verb form, orthography, noun number, morphology and determiners. They struggle with errors with conjugation, adjectives, adverbs and punctuation. Errors of the "other" type also seem to pose a challenge to all three models.

For comparison *between* the different models we will focus on only a few aspects of the results. Lets first do byt5 versus mt5. These models are pre-trained in the most similar way, and the comparison of these is our best way to study how the representation level (byte versus subword) affects prediction abilities. On the highest granularity level, mt5 struggles more than byt5 on Unnecessary words and phrases. On granularity level 2, we can see that mt5 performs better on only two categories: noun inflection and punctuation. For the two noise-related error types (morphology and spelling) the advantage of byt5 is smaller than one would expect from the difference in overall performance. From the lowest level of granularity we can highlight some of the results that stand out as different from what could be expected from overall performance. mt5 is good at errors related to missing pronouns, replacing of noun-inflections, replacing prepositions, and replacing punctuation. It is noticeably bad at correcting missing adverbs, conjugations, particles and verbs, as well as removing unnecessary adjectives and nouns.

Now we can turn to comparing byt5 to t5, where the overall result is

more comparable. Just a reminder that the overall $F_{0.5}$ scores were 0.504 and 0.510 respectively. For level 1 of granularity (figure 6.1a) we see that both models are strongest on Replacement and weakest on Unnecessary type edits. There is no significant difference between the models, with a 0.48/0.48, 0.53/0.52 and 0.41/0.40 relation for Missing, Replacements, and Unnecessary type edits for `t5`/`byt5` respectively. If anything, `byt5` is a bit weaker on correcting Replacement type edits than one would expect from the overall difference in performance. We can also note that Replacements account for 71 percent of the errors in the test data, so being best on this is important for the final score. As long as the distribution is the same in the training data, it could also explain why both models are best at this type of errors, as they have seen more examples of that type of error.

Level 2 (6.1b) of granularity focuses on the different parts-of-speech, while also including orthographical and punctuation categories. We ignore each category with less than 10 instances, and they are omitted from the figure. All scores are given in $F_{0.5}$. For `t5` the strongest categories are VERB:FORM 0.67, SPELL 0.82, NOUN:INFL 0.95. The corresponding numbers for `byt5` are 0.60, 0.80 and 0.95. The strongest scores for `byt5` are on NOUN:INFL 0.95, SPELL 0.80 and NOUN:NUM at 0.67. The biggest differences between the models are that `t5` is better on word order, adverbs, particles and subject-verb agreement, while `byt5` is significantly better at punctuation, and slightly better at adjectives and "other".

Level 3 (figures 6.2a, 6.2b and 6.2c) of granularity is the most precise categorization of error types, where part-of-speech (POS) and edit-type are combined. One example could be that M:ADJ means a missing-adjective error. In the strong end: The three strongest categories with edit-type + POS for `t5` (corresponding score for `byt5` in parenthesis) are R:NOUN:INFL 0.95 (0.95), R:SPELL 0.82 (0.80) and R:VERB:FORM 0.67 (0.60), while for `byt5` (`t5` in parenthesis) it is R:NOUN:INFL 0.95 (0.95), R:SPELL 0.80 (0.82) and M:VERB:FORM 0.71 (0.60). The similarity in results on these particular categories compared to level 2 is explained by the fact that change in inflection, spelling, or morphological changes are always considered to be of the type Replacement.

Now lets look at the biggest differences between the models on granularity level 3. `t5` is better at correcting missing conjugations, particles and pronouns, replacement errors in adverbs and particles, and unnecessary adverbs and prepositions. `byt5` is better on missing punctuation and verb forms, replacing adjectives and punctuation, and noticing unnecessary adjectives, nouns and punctuation.

6.1.1 Conclusions

For English GEC, `byt5` is significantly better than `mt5`, and on the same level as `t5`. In this section we compared the byte-level model to the subword-level models on a error-wise basis. Our hypothesis about byte-level models handling noisy data is not supported by the results for the ERRANT granular evaluation, as we did not see any specific patterns showing that byte-level models excel at correcting errors on spelling and morphology compared

to any of the two subword-level models. The scores on spelling and morphology are more or less identical between `byt5` and `t5`. The byte-level model is clearly better at fixing punctuation-related errors than `t5`, but this pattern does not hold when comparing to `mt5`. Some of the other sub-word level categories, like verb and noun form and tense, are in favor of `t5`.

6.2 Norwegian - fine-grained-ASKEVAL

To be able to do a more fine-grained error-analysis for Norwegian, we have created a new evaluation scheme: ASKEVAL. This evaluation is only done on the ASK-EXPANDED evaluation dataset, which is not the same dataset as the one we used for $F_{0.5}$ score comparisons in chapter 5. Since this evaluation dataset has a maximum of one error in each sentence, we can easily categorise the sentences by their ASK-annotated error type and get scores for each of the error types based on the comparison of whole sentences.

Since the expanded evaluation set gives us one error tag per sentence, we can use a comparison of predictions to targets directly to study the abilities of the models. There are four (tp, fp, fn, tn) possible outcomes of an evaluation. From these we calculate precision and recall, and finally $F_{0.5}$. This is done for each error type.

The reason why we made a custom evaluation, instead of using a pre-made Python Library, is that for GEC we need to include the source sentence in the comparison stage. Most Python libraries, for example `ScikitLearn`, compare only target with prediction when calculating precision, recall and $F_{0.5}/F_{1,0}$ scores. But for the particular case of source, target and prediction being equal, `ScikitLearn` would give true positives when we want true negatives. A copying of the input when the reference is the same should not contribute positively to a GEC models score, as that is a too easy task. Therefore we need access to the source sentence at evaluation time.

6.2.1 Caveats when using ASKEVAL

There are two disadvantages to this scheme. First, the evaluation happens on a partly artificial set of sentences. This does not match the final use case of a GEC system, which is to correct naturalistic sentences. Second, the comparison of whole sentences does not differentiate between predictions errors on the span in focus and random errors occurring other places in the sentence. Therefore a scoring of a sentence describes how well a model predicts sentences as a whole, and not necessarily corrections of the error in focus.

6.2.2 Result discussion

We present the results on ASKEVAL for `nort5`, `mt5` and `byt5` both as tables (6.4 and 6.5) and as a bar-chart (6.3).

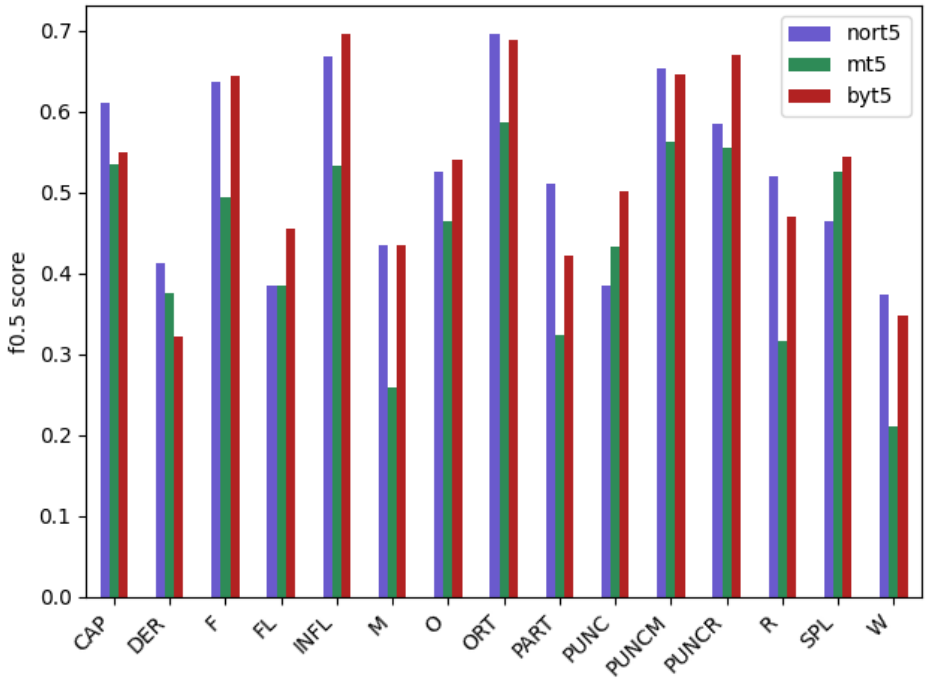


Figure 6.3: Comparing the performance of three models on the different error types in ASK. Round 7: Norwegian

In this section all the results are from round 7 in chapter 5, meaning it is from evaluation on unseen test-data, and with an adjusted learning rate for the `nort5` model. Before looking at the comparisons in figure 6.3, we again need to establish that `mt5` had an overall significantly lower score than the two other models. It outperforms any of the other models only on one error-type. All the comparisons with that model are therefore taking that into account, and if `mt5` is described as doing well on a error-type, this is only in relative terms.

First we can quickly summarize the patterns shown in figure 6.3 that are common to all three models. The strongest categories are errors related to letter casing, morphology, inflection, orthography, and removal and addition of punctuation. The models seem to struggle with errors with derivation, foreign word insertions, missing words and wrong word choices.

Now we can move to the variation between the three models. First we take the error-types where we expected the byte-level model to excel based on our hypothesis about noisy data: orthography (ORT) and morphology (F). `nort5` and `byt5` have very even performance on these categories.

Then to where the models differ. When comparing `byt5` to `mt5`, the byte-level model is around 0.1 better on most parameters. The exceptions are casing issues, derivatives and compound missing. There `mt5` performs comparably or better than `byt5`.

Then we compare `byt5` to `nort5`. `nort5` is better than `byt5` on casing (CAP), derivation (DER) and unexpected compounds (PART). The opposite

model	precision	recall	$F_{0.5}$
nort5	0.569	0.857	0.61
byt5	0.507	0.831	0.55
mt5	0.504	0.697	0.534

(a) Casing

model	precision	recall	$F_{0.5}$
nort5	0.38	0.633	0.413
byt5	0.333	0.282	0.321
mt5	0.394	0.317	0.376

(b) Derivation

model	precision	recall	$F_{0.5}$
nort5	0.614	0.75	0.637
byt5	0.621	0.747	0.643
mt5	0.507	0.449	0.494

(c) Morphology

model	precision	recall	$F_{0.5}$
nort5	0.333	1	0.384
byt5	0.5	0.333	0.454
mt5	0.375	0.429	0.385

(d) International words

model	precision	recall	$F_{0.5}$
nort5	0.624	0.918	0.667
byt5	0.673	0.8	0.695
mt5	0.521	0.581	0.532

(e) Wrong inflection for this word

model	precision	recall	$F_{0.5}$
nort5	0.427	0.464	0.434
byt5	0.432	0.441	0.434
mt5	0.283	0.193	0.259

(f) Missing word

model	precision	recall	$F_{0.5}$
nort5	0.49	0.741	0.526
byt5	0.509	0.717	0.54
mt5	0.451	0.525	0.464

(g) Word/phrase order

model	precision	recall	$F_{0.5}$
nort5	0.658	0.908	0.696
byt5	0.669	0.782	0.689
mt5	0.578	0.627	0.587

(h) Spelling

Figure 6.4: One table for each error type in ASK and how the different models handled those error types. The rest of the error types are in figure 6.5

is the case for the use of foreign words (FL), punctuation in general (PUNC), superfluous punctuation (PUNCR) and compound missing (SPL). We have found some examples (table 6.2) where the models differ on these parameters. In example 1, nort5 manages to correct the derivation error while byt5 does not. *Rolighet* is not a valid derivation of *ro* in Norwegian, but the pattern adj-to-noun exists with words like *god - godhet*. In example 2, the form *vennenskap* follows the exact same pattern as example 1, being a derivation error. This indicates that we cannot completely trust the error-codes of our augmented ASK corpus. In example 3 *alt for* should be written together, and in example 4, a comma could be argued to be correct.

So what conclusions can be drawn from this? Again, as for English, we see that the byte-level model is good at punctuation. Other than that there are no clear patterns in the differences between the models, and we did not find evidence to support that byte-level representation is advantageous on the so-called "noisy" error-types. In compounding t5 and byt5 behave oppositely for the two subcategories *unexpected compounding* and *compound missing*. That indicates that byt5 "prefers" to write words together when

model	precision	recall	$F_{0.5}$
nort5	0.5	0.56	0.511
byt5	0.458	0.318	0.421
mt5	0.339	0.276	0.324

(a) CompoundingPART

model	precision	recall	$F_{0.5}$
nort5	0.36	0.529	0.385
byt5	0.48	0.61	0.501
mt5	0.411	0.545	0.432

(b) Punctuation (general)

model	precision	recall	$F_{0.5}$
nort5	0.647	0.68	0.653
byt5	0.633	0.698	0.645
mt5	0.563	0.558	0.562

(c) Punctuation (missing)

model	precision	recall	$F_{0.5}$
nort5	0.6	0.531	0.585
byt5	0.683	0.617	0.669
mt5	0.586	0.456	0.554

(d) Punctuation (remove)

model	precision	recall	$F_{0.5}$
nort5	0.492	0.666	0.519
byt5	0.443	0.615	0.469
mt5	0.308	0.35	0.316

(e) Superfluous word

model	precision	recall	$F_{0.5}$
nort5	0.432	0.662	0.464
byt5	0.508	0.759	0.544
mt5	0.505	0.629	0.526

(f) Compounding - should be together

model	precision	recall	$F_{0.5}$
nort5	0.374	0.375	0.374
byt5	0.361	0.306	0.348
mt5	0.241	0.138	0.21

(g) Wrong word

Figure 6.5: One table for each error type in ASK, showing how the different models handled those error types. Listing precision, recall and $F_{0.5}$ score

unsure. mt5 follows this pattern of the byte-level model when it comes to compounding errors.

6.3 Norwegian - effect of data augmentation

The problem of data sparsity for GEC training was discussed in chapter 3 section 3.4.3. This problem has motivated a lot of research into data augmentation, also called synthetic data generation. We have made a simple augmentation of ASK, and the augmented datasets were tested for GEC model fine-tuning in chapter 5.

We will now look at the effect of these training data augmentations on final $F_{0.5}$ scores for our models, on both types of evaluation datasets. The results are shown in the bar charts of figure 6.6. These results are from round 6 of training (section 5.3.3), before the hyper-parameter adjustments for nort5 which significantly improved that models performance. This unfortunately means that the absolute results on that model should be interpreted with caution, as the results are unfairly weak. All the evaluation

Model	Sentence
	Example 1 - Derivation
Source	Det gir rolighet.
t5	**Det gir ro.**
byt5	Det gir rolighet.
	Example 2 - Unexpected compounding
Source	Uten kjærlighet og vennenskap kan man ikke leve.
t5	**Uten kjærlighet og vennskap kan man ikke leve.**
byt5	Uten kjærlighet og vennenskap kan man ikke leve.
	Example 3 - Missed compounding
Source	Nå for tiden sitter barn alt for mye inne.
t5	Nå for tiden sitter barn alt for mye inne.
byt5	**Nå for tiden sitter barn altfor mye inne.**
	Example 4 - Punctuation
Source	Å vise ansvar gir trygghet for andre mennesker.
t5	Å vise ansvar gir trygghet for andre mennesker.
byt5	**Å vise ansvar, gir trygghet for andre mennesker.**

Table 6.2: Some examples of when nort5 and byt5 differ in the predictions they output. Representative for the differences in error-types performance differences. Correct corrections are **highlighted**

is done on development data, not test data.

The two bar charts in figure 6.6 show the effect the training data used has on performance, $F_{0.5}$ score specifically. For the evaluation on raw data, nort5 and byt5 show the same pattern: using only expanded training data is worst, and using only the raw data is the best. Since using both datasets more than doubles the training cost, it is clear that using only the raw dataset is the best option. For mt5 the results are quite different, and using the expanded dataset is positive for its performance.

Looking at the second chart with evaluation on the one-error-per-sentence expanded evaluation set, it is clear that training on the expanded dataset is advantageous. It shows how training on the same data as we are evaluating on is an important factor.

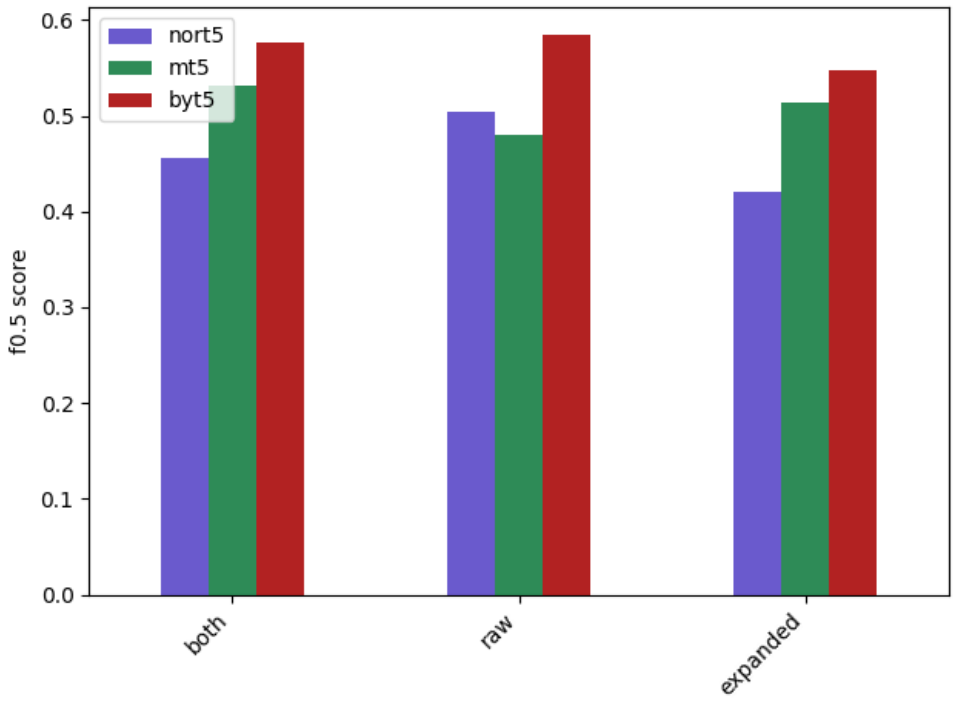
mt5 has a particular behaviour across the two evaluation schemes. Training solely on raw data seems to be detrimental for the model.

What can be the reason the expanded dataset did not increase performance the way other data augmentation techniques have been shown to do? First of all, our technique does not add any new vocabulary to the training data. Seeing more vocabulary, and how that vocabulary behaves when it comes to GEC, could give a model an ability to correct a larger variation of errors. Many other data augmentation techniques add new vocabulary to the data, since they are using new (clean) sources of text. Our

method does not add any new errors either, and only repeats the already existing errors in slightly new contexts. These new contexts are the same sentences, but without any additional errors. This variation in context did not seem to be enough to give the models better performance.

To conclude, we can say that the ASK-EXPANDED dataset is only useful for fine-tuning the `mt5` model. Using both datasets together is never better than only using one of them, especially when considering the increased fine-tuning cost. Other data augmentation methods have to be tried out for Norwegian to significantly improve performance on GEC.

a.)



b.)

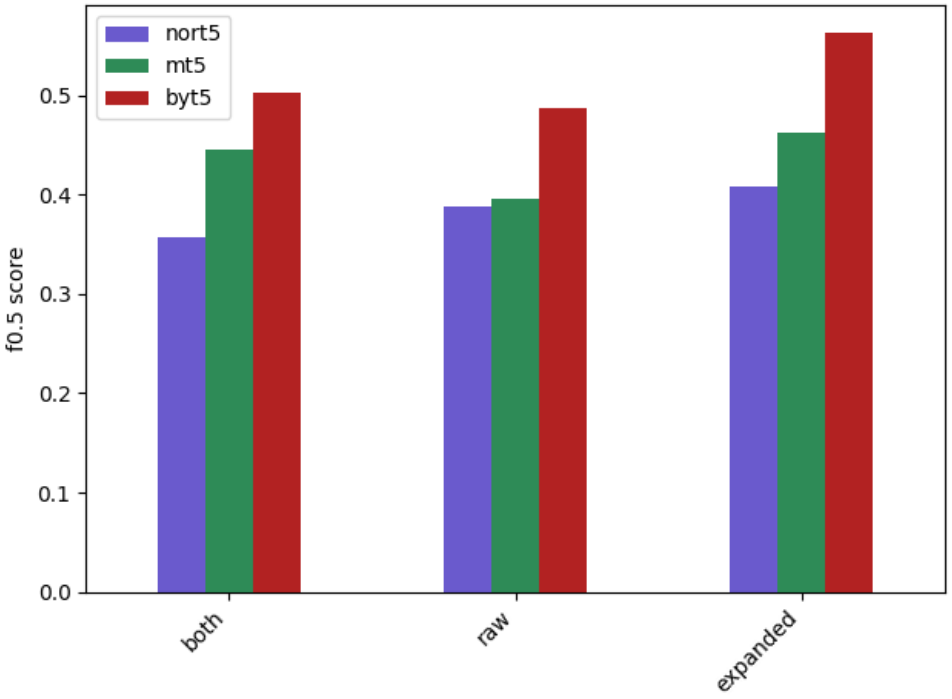


Figure 6.6: Exploring the effect of different training data (x-axis) on GEC performance. Three models. a.) is evaluation on ASK-RAW, b.) is evaluation on ASK-EXPANDED.

Chapter 7

Conclusion

7.1 Byte level representation on English GEC

In this thesis we have tried out byte-level language representation for grammatical error correction models. We have experimented with various hyperparameters, and achieved a $F_{0.5}$ score of 0.51 for English with our best model, with training and testing on the CLC-FCE dataset. We used the ERRANT scorer (Bryant et al., 2017) to evaluate our models.

We compared the performance of the multilingual byte-level `byt5` model (Xue et al., 2022) with two subword-level models. First with the `t5` model (Raffel et al., 2020), which is pre-trained mostly on English data. Second with `byt5`s subword-level, multilingual counterpart `mt5` (Xue et al., 2021). For English, the subword-level `t5` model slightly over-performed the byte-level model, with $F_{0.5}$ scores of 0.51 and 0.504 respectively. Compared to the multilingual `mt5` however ($F_{0.5}$ 0.428), the byte-level representation was clearly better. This shows that byte-level representation can be a good option when training GEC models for English.

We have also compared the performances between these models on an error-type level. Our hypothesis that byte-level representation might help with noisy data like orthographical and morphological errors did not hold water. But there were some differences arising from difference in representation level, like the fact that byte-level models performed better on punctuation-related errors, but worse on particle-, word order- and agreement-related errors.

In addition to comparing performance on GEC, we have looked at some advantages and disadvantages of using byte level representation. We have seen that our byte-level models took almost three times as long to develop due to longer sequences, which is a disadvantage. The advantage of byte-level models is that all decisions regarding tokenization are by-passed, and these kind of models can be pre-trained almost "out-of-the-box". The fact that subword-levels need a tokenization step is not a great problem for English, as the current methods for doing this in English yield results that compare with without-tokenizer systems when measured in $F_{0.5}$. For the researcher who has to decide what language representation level to use for training a GEC model for English, it comes down to whether to

emphasize quicker training time subword models, or better performance on punctuation errors and less pre-processing with byte-level models.

7.2 GEC for Norwegian

With the help of the ASK dataset (Tenfjord et al., 2006) we have fine-tuned several pre-trained models that can now perform grammatical error correction in Norwegian. We fine-tuned models built with both subword- and byte-level text representation. We have established a baseline for GEC on Norwegian, and achieved a $F_{0.5}$ score of 0.581 with our best model. Like in modern NLP in general, the key to perform this task was a suitable dataset.

The multilingual byte-level model ($F_{0.5}$ score 0.581) clearly outperforms its subword-level, multilingual, counterpart `mt5` ($F_{0.5}$ 0.479), just as for English. It is performing just as well as, and even slightly better than, the subword-level model `nort5` ($F_{0.5}$ 0.577) (Samuel et al., 2023), which was pre-trained only on Norwegian texts. This raises the question whether language-specific pre-training is necessary for a task like GEC when one has access to byte-level multilingual models.

We created our own evaluation scheme for Norwegian GEC, so that we could study the error-wise performance of `byt5` compared to subword models. We primarily compared its performance to `nort5`, as the overall performance of `mt5` was not very good. As was the case for English, we did not find that byte-level representation helped with performance on what we expected to be typical noise-infused error types. The model did however excel at punctuation related errors, and errors related to wrong use of foreign words. It was a bit weaker than `nort5` on derivation-, letter casing-, and some compounds-related errors.

For Norwegian, we tested out a new scheme for data augmentation, where we split the original parallel dataset into one-error-each sentences, thereby increasing the size of the dataset. Training on this dataset did not increase performance of our main models, but was beneficial for the multilingual sub-word based `mt5`. The latter model did not however compare well with the other models on overall performance.

The Norwegian models we fine-tuned on the ASK dataset are made accessible for everyone through HuggingFace Hub.[1] With intermediate knowledge of PyTorch, one can use these models to perform grammatical error correction on an possibly erroneous input sentence. We hope that the work done in this thesis will be of help for anyone wanting to continue work on GEC for Norwegian and other low and medium resource languages. The `byt5` and `nort5` models we have fine-tuned have pretty similar results. When it comes to the decision on whether to use byte-level or subword-level for fine-tuning, it boils down to a priority of higher performance on certain error-types like punctuation and less pre-processing for `byt5`, and faster fine-tuning time and advantage on some other error-types with `nort5`.

[1] https://huggingface.co/MatiasJ

Chapter 8

Suggestions for future work

The study of GEC for English is a huge field with many researchers engaged in it, and there are still several challenges before the task is completely solved. In this thesis we have focused on a small part of the process of model development for English, and several other parts have remained untouched. For Norwegian, the GEC slate is mostly clean, and we have created a baseline for others to improve on. Some suggestions for possible system improvement and future research are discussed in the sections below.

8.1 More GEC training data for Norwegian

As we have only tested one class of training architectures for developing GEC for Norwegian, namely utilizing the `t5` family of pre trained seq2seq models, a wide landscape of possible approaches remain to be studied and tested.

Training of GEC systems with modern neural methods requires parallel corpora of non-corrected and corrected sentences. Creating such corpora is resource consuming, so there is a limited amount of training data for GEC. A focus of many recent approaches in GEC for English is to bypass the training data sparsity problem by using artificially generated datasets for model pre-training (see chapter 3 for more on this subject). The methods used for English could be applied in a Norwegian context as well.

One way of creating more training data is to insert errors into monolingual, error-free text. A possible source for such clean text is a Norwegian Wikipedia dump. Then one can introduce artificial errors to the corpora according to some set of rules.

Another method for creating more data is back-translation. A seq2seq model is trained in a "mirrored" fashion to what we have done when creating our GEC system. One could use the ASK GEC corpus, but the source and target swap places. Thereby we train a model to automatically corrupt clean text by adding errors. The resulting model can then be used on the sentences in a clean corpus like a Wikipedia dump. The output of the model become the source sentences in the new corpus, and the input becomes the target.

We are curious to see how much adding artificial pre-training data could improve GEC performance for Norwegian.

8.2 GEC evaluation for Norwegian

In this thesis we have had a limited set of tools for optimally evaluating the models we have created. For future work on GEC for Norwegian, a common evaluation framework would be beneficial. Improving evaluation of Norwegian GEC could involve adopting the ERRANT scorer for Norwegian. This adaptation has been done for several other languages already (see chapter 4). ERRANT uses linguistically motivated rules to perform alignment between source, target and prediction, and to categorize the error types according to properties of the text. These rules are facilitated by language-specific pre-processing steps like part-of-speech tagging and lemmatization. A language-specific word list is also needed for recognizing wrongly spelled words. The Czech adaptation of ERRANT (Náplava et al., 2022) relies on UDPipe, which is also available for Norwegian (Straka, 2020).

8.3 Language specific byte-level models

The byte-level models we used in our experiments, `byt5`, were pre-trained on data from over 100 languages, including English and Norwegian. These models turned out to be roughly equal in predictive power compared to the language-specific subword-level models when fine-tuned for GEC. What would happen if a byte-level model was trained on data only from one particular language? Would it just affect the number of fine-tuning steps needed to achieve the same performance as we did in our experiments? Or would it improve performance overall?

Intuitively, language-specificity sounds like a good idea. With the subword-level models, there is a significant improvement in GEC-performance when a model is pre-trained on language specific data. But we have to remember that the subword-level models depend on a language specific tokenization step for text representation, and the byte-level models do not have any such step. Perhaps some of the advantage of the subword-level models comes from the language-specific tokenization step, and the multilinguality might not be such a big issue for multilingual `byt5`. If this is true, a language-specific byte-level model might only give an improvement in terms of saved training time.

Empirical evidence is needed to answer this question, and we encourage future researchers to experiment with language-specific byte-level models.

8.4 GEC and society

In this thesis we have limited our considerations regarding GEC to the conversion of source sentences to predictions that match a target sentence. But what are the consequences when this task is possibly solved some time

in the future? Can school teachers be completely freed from correcting learner texts? Should we stop teaching correct writing to children, as they can just rely on a automatic system guessing what they tried to write? Will every writing software automatically correct everything we write in a digital format? All of these questions require knowledge from fields such as pedagogy, psychology and sociology. Society will undoubtedly be faced with these questions at some point, and political decisions will have to be made regarding them.

Bibliography

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *ArXiv*, *1409*.

Behera, B., & Bhattacharyya, P. (2013). Automated grammar correction using hierarchical phrase-based statistical machine translation.

Belkebir, R., & Habash, N. (2021). Automatic error type annotation for Arabic. *Proceedings of the 25th Conference on Computational Natural Language Learning*, 596–606. https://doi.org/10.18653/v1/2021.conll-1.47

Berggren, S. J. (2019). Automated assessment of norwegian l2 essays using multi-task learning. master thesis, university of oslo.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: Analyzing text with the natural language toolkit*. O'Reilly. https://doi.org/http://my.safaribooksonline.com/9780596516499

Boyd, A. (2018). Using wikipedia edits in low resource grammatical error correction. *Proceedings of the 4th Workshop on Noisy User-generated Text*. http://aclweb.org/anthology/W18-6111

Brockett, C., Dolan, W., & Gamon, M. (2006). Correcting esl errors using phrasal smt techniques. *Proc. ACL*. https://doi.org/10.3115/1220175.1220207

Bryant, C., Felice, M., Andersen, Ø. E., & Briscoe, T. (2019). The BEA-2019 shared task on grammatical error correction. *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 52–75. https://doi.org/10.18653/v1/W19-4406

Bryant, C., Felice, M., & Briscoe, T. (2017). Automatic annotation and evaluation of error types for grammatical error correction. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 793–805. https://doi.org/10.18653/v1/P17-1074

Bryant, C., Yuan, Z., Qorib, M. R., Cao, H., Ng, H. T., & Briscoe, T. (2023). Grammatical error correction: A survey of the state of the art.

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111. https://doi.org/10.3115/v1/W14-4012

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation.

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1724–1734. https://doi.org/10.3115/v1/D14-1179

Chollampatt, S., & Ng, H. T. (2017). Connecting the dots: Towards human-level grammatical error correction. *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, 327–333. https://doi.org/10.18653/v1/W17-5037

Chollampatt, S., Wang, W., & Ng, H. T. (2019). Cross-sentence grammatical error correction. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 435–445. https://doi.org/10.18653/v1/P19-1042

Chomsky, N. (1965). *Aspects of the theory of syntax*. The MIT Press. http://www.amazon.com/Aspects-Theory-Syntax-Noam-Chomsky/dp/0262530074

Choshen, L., & Abend, O. (2018). Reference-less measure of faithfulness for grammatical error correction. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 124–129. https://doi.org/10.18653/v1/N18-2020

Dahlmeier, D., & Ng, H. T. (2012). Better evaluation for grammatical error correction. *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 568–572. https://aclanthology.org/N12-1067

Dahlmeier, D., Ng, H. T., & Wu, S. M. (2013). Building a large annotated corpus of learner English: The NUS corpus of learner English. *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, 22–31. https://aclanthology.org/W13-1703

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *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)*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Fabbri, A. R., Kryściński, W., McCann, B., Xiong, C., Socher, R., & Radev, D. (2021). SummEval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, *9*, 391–409. https://doi.org/10.1162/tacl_a_00373

Gamon, M., Leacock, C., Brockett, C., William B Dolan, J. G., Belenko, D., & Klementiev, A. (2009). Using statistical techniques and web search to correct esl errors.

Grundkiewicz, R., & Junczys-Dowmunt, M. (2018). Near human-level performance in grammatical error correction with hybrid machine translation. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 284–290. https://doi.org/10.18653/v1/N18-2046

Grundkiewicz, R., Junczys-Dowmunt, M., & Heafield, K. (2019). Neural grammatical error correction systems with unsupervised pre-training on synthetic data. *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 252–263. https://doi.org/10.18653/v1/W19-4427

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, *abs/1207.0580*. http://arxiv.org/abs/1207.0580

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing* [To appear].

Hu, J., Ruder, S., Siddhant, A., Neubig, G., Firat, O., & Johnson, M. (2020). XTREME: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. *CoRR*, *abs/2003.11080*. https://arxiv.org/abs/2003.11080

Jentoft, M., & Samuel, D. (2023). NocoLA: The norwegian corpus of linguistic acceptability. *The 24rd Nordic Conference on Computational Linguistics*. https://openreview.net/forum?id=UcWZrerHDCe

Junczys-Dowmunt, M., Grundkiewicz, R., Guha, S., & Heafield, K. (2018). Approaching neural grammatical error correction as a low-resource machine translation task. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 595–606. https://doi.org/10.18653/v1/N18-1055

Kementchedjhieva, Y., & Søgaard, A. (2023). Grammatical error correction through round-trip machine translation. *Findings of the Association for Computational Linguistics: EACL 2023*, 2208–2215. https://aclanthology.org/2023.findings-eacl.165

Kiyono, S., Suzuki, J., Mita, M., Mizumoto, T., & Inui, K. (2019). An empirical study of incorporating pseudo data into grammatical error correction. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 1236–1242. https://doi.org/10.18653/v1/D19-1119

Kozodoi, N. (2021). Gradient accumulation in pytorch. https://kozodoi.me/python/deep%5C%20learning/pytorch/tutorial/2021/02/19/gradient-accumulation.html

Kudo, T., & Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *CoRR*, *abs/1808.06226*. http://arxiv.org/abs/1808.06226

Kummervold, P. E., De la Rosa, J., Wetjen, F., & Brygfjeld, S. A. (2021). Operationalizing a national digital library: The case for a Norwegian transformer model. *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, 20–29. https://aclanthology.org/2021.nodalida-main.3

Kutuzov, A., Barnes, J., Velldal, E., Øvrelid, L., & Oepen, S. (2021). Large-scale contextualised language modelling for Norwegian. *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, 30–40. https://aclanthology.org/2021.nodalida-main.4

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

Li, R., Wang, C., Zha, Y., Yu, Y., Guo, S., Wang, Q., Liu, Y., & Lin, H. (2019). The LAIX systems in the BEA-2019 GEC shared task. *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 159–167. https://doi.org/10.18653/v1/W19-4416

Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., & Tan, S. (2021). Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *CoRR*, *abs/2112.10508*. https://arxiv.org/abs/2112.10508

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, *2013*.

Naber, D. (2003). A rule-based style and grammar checker.

Náplava, J., Straka, M., Straková, J., & Rosen, A. (2022). Czech grammar error correction with a large and diverse corpus. *Transactions of the Association for Computational Linguistics*, *10*, 452–467. https://doi.org/10.1162/tacl_a_00470

Napoles, C., Sakaguchi, K., Post, M., & Tetreault, J. (2015). Ground truth for grammatical error correction metrics. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 588–593. https://doi.org/10.3115/v1/P15-2097

Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., & Bryant, C. (2014). The CoNLL-2014 shared task on grammatical error correction. *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, 1–14. https://doi.org/10.3115/v1/W14-1701

Nicholls, D. (2003). The cambridge learner corpus - error coding and analysis for lexicography and elt.

Omelianchuk, K., Atrasevych, V., Chernodub, A., & Skurzhanskyi, O. (2020). GECToR – grammatical error correction: Tag, not rewrite. *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, 163–170. https://doi.org/10.18653/v1/2020.bea-1.16

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318. https://doi.org/10.3115/1073083.1073135

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B.,

Fang, L., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2227–2237. https://doi.org/10.18653/v1/N18-1202

Popel, M., & Bojar, O. (2018). Training tips for the transformer model. *CoRR, abs/1804.00247*. http://arxiv.org/abs/1804.00247

Povlsen, C., Hein, A. S., de Smedt, K., Löfvendahl, B., Paggio, P., Persson, O., & Rosén, V. (1999). Final project report, scarrie.

Qorib, M., Na, S.-H., & Ng, H. T. (2022). Frustratingly easy system combination for grammatical error correction. *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1964–1974. https://doi.org/10.18653/v1/2022.naacl-main.143

Qorib, M. R., & Ng, H. T. (2022). Grammatical error correction: Are we there yet? *Proceedings of the 29th International Conference on Computational Linguistics*, 2794–2800. https://aclanthology.org/2022.coling-1.246

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*(140), 1–67. http://jmlr.org/papers/v21/20-074.html

Rothe, S., Mallinson, J., Malmi, E., Krause, S., & Severyn, A. (2021). A simple recipe for multilingual grammatical error correction. *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)*, 702–707. https://doi.org/10.18653/v1/2021.acl-short.89

Samuel, D., Kutuzov, A., Touileb, S., Velldal, E., Øvrelid, L., Rønningstad, E., Sigdel, E., & Palatkina, A. S. (2023). Norbench – a benchmark for norwegian language models. *The 24rd Nordic Conference on Computational Linguistics*. https://openreview.net/forum?id=WgxNONkAbz

Samuel, D., & Straka, M. (2021). ÚFAL at MultiLexNorm 2021: Improving multilingual lexical normalization by fine-tuning ByT5. *Proceedings of the 7th Workshop on Noisy User-generated Text (W-NUT 2021)*.

Shah, D. S., Schwartz, H. A., & Hovy, D. (2020). Predictive biases in natural language processing models: A conceptual framework and overview. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 5248–5264. https://doi.org/10.18653/v1/2020.acl-main.468

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Straka, M. (2020). Udpipe norwegian bokmaal: Morphosyntactic analysis of raw text. version 1.2.1-ud2.4. https://doi.org/10.57771/hvn0-kb58

Tajiri, T., Komachi, M., & Matsumoto, Y. (2012). Tense and aspect error correction for ESL learners using global context. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 198–202. https://aclanthology.org/P12-2039

Tarnavskyi, M., Chernodub, A., & Omelianchuk, K. (2022). Ensembling and knowledge distilling of large sequence taggers for grammatical error correction. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3842–3852. https://aclanthology.org/2022.acl-long.266

Tenfjord, K., Meurer, P., & Hofland, K. (2006). *The ASK Corpus – A Language Learner Corpus of Norwegian as a Second Language. Proceedings from 5th International Conference on Language Resources and Evaluation (LREC), Genova 2006.* http://www.lrec-conf.org/proceedings/lrec2006/pdf/573%5C_pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. http://arxiv.org/abs/1706.03762

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 353–355. https://doi.org/10.18653/v1/W18-5446

Wang, Y., Wang, Y., Dang, K., Liu, J., & Liu, Z. (2021). A comprehensive survey of grammatical error correction. *ACM Trans. Intell. Syst. Technol.*, *12*(5). https://doi.org/10.1145/3474840

Warstadt, A., Singh, A., & Bowman, S. R. (2019). Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, *7*, 625–641. https://doi.org/10.1162/tacl_a_00290

Winata, G. I., Madotto, A., Lin, Z., Liu, R., Yosinski, J., & Fung, P. (2021). Language models are few-shot multilingual learners. *Proceedings of the 1st Workshop on Multilingual Representation Learning*, 1–15. https://doi.org/10.18653/v1/2021.mrl-1.1

Winder, R. V. P., MacKinnon, J., Li, S. Y., Lin, B. C. T. L., Heah, C. L. H., Morgado da Costa, L., Kuribayashi, T., & Bond, F. (2017). NTUCLE: Developing a corpus of learner English to provide writing support for engineering students. *Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017)*, 1–11. https://aclanthology.org/W17-5901

Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., & Ng, A. (2016). Neural language correction with character-based attention.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., & Raffel, C. (2022). ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for*

Computational Linguistics, *10*, 291–306. https://doi.org/10.1162/tacl_a_00461

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2021). MT5: A massively multilingual pre-trained text-to-text transformer. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 483–498. https://doi.org/10.18653/v1/2021.naacl-main.41

Yannakoudakis, H., Briscoe, T., & Medlock, B. (2011). A New Dataset and Method for Automatically Grading ESOL Texts. *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Yuan, Z., & Briscoe, T. (2016). Grammatical error correction using neural machine translation. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 380–386. https://doi.org/10.18653/v1/N16-1042

Zaghouani, W., Mohit, B., Habash, N., Obeid, O., Tomeh, N., Rozovskaya, A., Farra, N., Alkuhlani, S., & Oflazer, K. (2014). Large scale arabic error annotation: Guidelines and framework. https://doi.org/10.13140/RG.2.1.3273.6169

Appendix

8.5 Error wise results round 5

On the following pages are the full evaluations from the ERRANT automatic scorer on the experiments in round 5 from section 5.2.6. The first pdf is from the predictions of the GEC-trained `t5` model, the second for `mt5`, and the third is for the GEC-trained `byt5` model. The numbers in these tables are used to create the bar-charts in figures 6.1 and 6.2.

```
Granular evaluation for round5 - T5
Run identifier: number3

Top level evaluation
=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
2284    1742    3988    0.5673  0.3642  0.5104
==============================================

Granularity level 1
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M               353     304     691     0.5373  0.3381  0.4807
R               1729    1182    2851    0.594   0.3775  0.5329
U               202     256     446     0.441   0.3117  0.4073

Granularity level 2
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
ADJ             26      46      126     0.3611  0.1711  0.2955
ADJ:FORM        3       1       2       0.75    0.6     0.7143
ADV             23      28      123     0.451   0.1575  0.3286
CONJ            1       7       27      0.125   0.0357  0.0833
CONTR           1       3       4       0.25    0.2     0.2381
DET             316     192     346     0.622   0.4773  0.5865
MORPH           55      27      50      0.6707  0.5238  0.6351
NOUN            224     227     598     0.4967  0.2725  0.4265
NOUN:INFL       16      1       0       0.9412  1.0     0.9524
NOUN:NUM        62      32      57      0.6596  0.521   0.6263
NOUN:POSS       1       0       0       1.0     1.0     1.0
ORTH            99      53      78      0.6513  0.5593  0.6306
OTHER           214     448     1016    0.3233  0.174   0.2759
PART            19      15      34      0.5588  0.3585  0.5026
PREP            315     204     385     0.6069  0.45    0.5674
PRON            104     54      130     0.6582  0.4444  0.6005
PUNCT           6       22      46      0.2143  0.1154  0.1829
SPELL           378     67      138     0.8494  0.7326  0.8232
VERB            117     150     412     0.4382  0.2212  0.3663
VERB:FORM       106     48      66      0.6883  0.6163  0.6726
VERB:INFL       7       1       0       0.875   1.0     0.8974
VERB:SVA        40      25      29      0.6154  0.5797  0.6079
VERB:TENSE      123     84      265     0.5942  0.317   0.5058
WO              28      7       56      0.8     0.3333  0.625

Granularity level 3
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           6       8       25      0.4286  0.1935  0.3448
M:ADV           4       6       26      0.4     0.1333  0.2857
M:CONJ          1       3       13      0.25    0.0714  0.1667
```

M:DET	175	99	144	0.6387	0.5486	0.6184
M:NOUN	8	19	59	0.2963	0.1194	0.2286
M:OTHER	7	38	167	0.1556	0.0402	0.0989
M:PART	2	3	10	0.4	0.1667	0.3125
M:PREP	64	46	87	0.5818	0.4238	0.5415
M:PRON	42	26	43	0.6176	0.4941	0.5882
M:PUNCT	1	7	19	0.125	0.05	0.0962
M:VERB	16	33	52	0.3265	0.2353	0.303
M:VERB:FORM	6	4	4	0.6	0.6	0.6
M:VERB:TENSE	21	12	42	0.6364	0.3333	0.5385
R:ADJ	19	33	82	0.3654	0.1881	0.3074
R:ADJ:FORM	3	1	2	0.75	0.6	0.7143
R:ADV	14	16	59	0.4667	0.1918	0.3627
R:CONJ	0	2	9	0.0	0.0	0.0
R:CONTR	0	2	1	0.0	0.0	0.0
R:DET	63	45	123	0.5833	0.3387	0.5097
R:MORPH	55	27	50	0.6707	0.5238	0.6351
R:NOUN	210	189	494	0.5263	0.2983	0.4565
R:NOUN:INFL	16	1	0	0.9412	1.0	0.9524
R:NOUN:NUM	62	32	57	0.6596	0.521	0.6263
R:NOUN:POSS	1	0	0	1.0	1.0	1.0
R:ORTH	99	53	78	0.6513	0.5593	0.6306
R:OTHER	195	305	775	0.39	0.201	0.3283
R:PART	14	12	17	0.5385	0.4516	0.5185
R:PREP	196	130	220	0.6012	0.4712	0.5698
R:PRON	48	18	62	0.7273	0.4364	0.6417
R:PUNCT	5	9	12	0.3571	0.2941	0.3425
R:SPELL	378	67	138	0.8494	0.7326	0.8232
R:VERB	95	101	331	0.4847	0.223	0.3926
R:VERB:FORM	96	43	60	0.6906	0.6154	0.6742
R:VERB:INFL	7	1	0	0.875	1.0	0.8974
R:VERB:SVA	40	25	29	0.6154	0.5797	0.6079
R:VERB:TENSE	85	63	196	0.5743	0.3025	0.4868
R:WO	28	7	56	0.8	0.3333	0.625
U:ADJ	1	5	19	0.1667	0.05	0.1136
U:ADV	5	6	38	0.4545	0.1163	0.2874
U:CONJ	0	2	5	0.0	0.0	0.0
U:CONTR	1	1	3	0.5	0.25	0.4167
U:DET	78	48	79	0.619	0.4968	0.59
U:NOUN	6	19	45	0.24	0.1176	0.1987
U:OTHER	12	105	74	0.1026	0.1395	0.1083
U:PART	3	0	7	1.0	0.3	0.6818
U:PREP	55	28	78	0.6627	0.4135	0.5914
U:PRON	14	10	25	0.5833	0.359	0.5185
U:PUNCT	0	6	15	0.0	0.0	0.0
U:VERB	6	16	29	0.2727	0.1714	0.2439
U:VERB:FORM	4	1	2	0.8	0.6667	0.7692
U:VERB:TENSE	17	9	27	0.6538	0.3864	0.5743

Figure 8.1: The full ERRANT fine-grained, error-wise, evaluation from round 5. Model: `t5`

```
mt5
Granular evaluation for round 5 - mt5
Run identifier: number1-mt5_x

Top level evaluation
========== Span-Based Correction ============
TP      FP      FN      Prec            Rec     F0.5
1450    1211    4828    0.5449  0.231   0.4284
=============================================

Granularity level 1
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M               190     110     858     0.6333  0.1813  0.4226
R               1141    791     3438    0.5906  0.2492  0.4636
U               119     310     532     0.2774  0.1828  0.2514

Granularity level 2
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
ADJ             21      29      132     0.42    0.1373  0.2975
ADJ:FORM        1       0       4       1.0     0.2     0.5556
ADV             7       24      139     0.2258  0.0479  0.1296
CONJ            0       2       28      0.0     0.0     0.0
CONTR           2       1       3       0.6667  0.4     0.5882
DET             177     114     485     0.6082  0.2674  0.4847
MORPH           43      22      61      0.6615  0.4135  0.5907
NOUN            134     188     692     0.4161  0.1622  0.3169
NOUN:INFL       16      0       0       1.0     1.0     1.0
NOUN:NUM        35      23      84      0.6034  0.2941  0.4986
NOUN:POSS       1       0       0       1.0     1.0     1.0
ORTH            72      67      103     0.518   0.4114  0.4925
OTHER           116     320     1112    0.2661  0.0945  0.1952
PART            6       10      47      0.375   0.1132  0.2564
PREP            201     105     499     0.6569  0.2871  0.5223
PRON            58      29      176     0.6667  0.2479  0.4983
PUNCT           8       8       49      0.5     0.1404  0.3306
SPELL           331     79      185     0.8073  0.6415  0.7676
VERB            57      68      473     0.456   0.1075  0.2767
VERB:FORM       71      54      102     0.568   0.4104  0.5275
VERB:INFL       2       0       5       1.0     0.2857  0.6667
VERB:SVA        25      24      44      0.5102  0.3623  0.4717
VERB:TENSE      52      37      336     0.5843  0.134   0.3495
WO              14      7       69      0.6667  0.1687  0.4192

Granularity level 3
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           2       1       30      0.6667  0.0625  0.2273
M:ADV           1       3       29      0.25    0.0333  0.1087
```

M:CONJ	0	0	14	1.0	0.0	0.0
M:DET	94	46	225	0.6714	0.2947	0.5347
M:NOUN	2	12	65	0.1429	0.0299	0.0813
M:OTHER	1	5	174	0.1667	0.0057	0.0251
M:PART	0	0	12	1.0	0.0	0.0
M:PREP	47	14	104	0.7705	0.3113	0.5949
M:PRON	26	16	59	0.619	0.3059	0.5138
M:PUNCT	1	0	21	1.0	0.0455	0.1923
M:VERB	3	6	65	0.3333	0.0441	0.1442
M:VERB:FORM	5	5	5	0.5	0.5	0.5
M:VERB:TENSE	8	2	55	0.8	0.127	0.3883
R:ADJ	19	21	82	0.475	0.1881	0.364
R:ADJ:FORM	1	0	4	1.0	0.2	0.5556
R:ADV	4	11	69	0.2667	0.0548	0.1504
R:CONJ	0	1	9	0.0	0.0	0.0
R:CONTR	0	0	1	1.0	0.0	0.0
R:DET	24	13	162	0.6486	0.129	0.3593
R:MORPH	43	22	61	0.6615	0.4135	0.5907
R:NOUN	131	153	576	0.4613	0.1853	0.3554
R:NOUN:INFL	16	0	0	1.0	1.0	1.0
R:NOUN:NUM	35	23	84	0.6034	0.2941	0.4986
R:NOUN:POSS	1	0	0	1.0	1.0	1.0
R:ORTH	72	67	103	0.518	0.4114	0.4925
R:OTHER	112	162	855	0.4088	0.1158	0.2714
R:PART	6	9	25	0.4	0.1935	0.3297
R:PREP	125	63	291	0.6649	0.3005	0.5351
R:PRON	25	8	85	0.7576	0.2273	0.5165
R:PUNCT	6	4	12	0.6	0.3333	0.5172
R:SPELL	331	79	185	0.8073	0.6415	0.7676
R:VERB	51	52	376	0.4951	0.1194	0.3039
R:VERB:FORM	63	45	94	0.5833	0.4013	0.5348
R:VERB:INFL	2	0	5	1.0	0.2857	0.6667
R:VERB:SVA	25	24	44	0.5102	0.3623	0.4717
R:VERB:TENSE	35	27	246	0.5645	0.1246	0.3308
R:WO	14	7	69	0.6667	0.1687	0.4192
U:ADJ	0	7	20	0.0	0.0	0.0
U:ADV	2	10	41	0.1667	0.0465	0.1099
U:CONJ	0	1	5	0.0	0.0	0.0
U:CONTR	2	1	2	0.6667	0.5	0.625
U:DET	59	55	98	0.5175	0.3758	0.4812
U:NOUN	1	23	51	0.0417	0.0192	0.0338
U:OTHER	3	153	83	0.0192	0.0349	0.0211
U:PART	0	1	10	0.0	0.0	0.0
U:PREP	29	28	104	0.5088	0.218	0.4017
U:PRON	7	5	32	0.5833	0.1795	0.4023
U:PUNCT	1	4	16	0.2	0.0588	0.1351
U:VERB	3	10	32	0.2308	0.0857	0.1724
U:VERB:FORM	3	4	3	0.4286	0.5	0.4412
U:VERB:TENSE	9	8	35	0.5294	0.2045	0.4018

Figure 8.2: The full ERRANT fine-grained, error-wise, evaluation from round 5. Model: `mt5`

```
Granular evaluation for round 5 - BYT5
Run identifier: number4

Top level evaluation
=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
2152    1627    4126    0.5695  0.3428  0.5029
==============================================

Granularity level 1
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M               332     276     716     0.5461  0.3168  0.477
R               1646    1150    2933    0.5887  0.3595  0.5221
U               174     201     477     0.464   0.2673  0.4045

Granularity level 2
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
ADJ             31      38      122     0.4493  0.2026  0.3613
ADJ:FORM        3       0       2       1.0     0.6     0.8824
ADV             15      37      131     0.2885  0.1027  0.2119
CONJ            1       11      27      0.0833  0.0357  0.0658
CONTR           3       1       2       0.75    0.6     0.7143
DET             299     195     363     0.6053  0.4517  0.5667
MORPH           60      35      44      0.6316  0.5769  0.6198
NOUN            207     194     619     0.5162  0.2506  0.4259
NOUN:INFL       16      1       0       0.9412  1.0     0.9524
NOUN:NUM        71      31      48      0.6961  0.5966  0.6736
NOUN:POSS       1       0       0       1.0     1.0     1.0
ORTH            100     49      75      0.6711  0.5714  0.6485
OTHER           214     327     1014    0.3956  0.1743  0.3154
PART            12      14      41      0.4615  0.2264  0.3822
PREP            286     221     414     0.5641  0.4086  0.5242
PRON            88      60      146     0.5946  0.3761  0.5327
PUNCT           11      17      46      0.3929  0.193   0.3254
SPELL           364     74      152     0.8311  0.7054  0.8025
VERB            101     132     429     0.4335  0.1906  0.3454
VERB:FORM       96      61      77      0.6115  0.5549  0.5993
VERB:INFL       7       1       0       0.875   1.0     0.8974
VERB:SVA        35      34      34      0.5072  0.5072  0.5072
VERB:TENSE      104     80      284     0.5652  0.268   0.4626
WO              27      14      56      0.6585  0.3253  0.5466

Granularity level 3
==================== Span-Based Correction ======================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           5       5       27      0.5     0.1562  0.3472
M:ADV           4       4       26      0.5     0.1333  0.3226
M:CONJ          1       8       13      0.1111  0.0714  0.1
```

M:DET	171	99	148	0.6333	0.5361	0.6112
M:NOUN	5	18	62	0.2174	0.0746	0.1572
M:OTHER	6	30	169	0.1667	0.0343	0.094
M:PART	1	1	11	0.5	0.0833	0.25
M:PREP	63	34	88	0.6495	0.4172	0.5844
M:PRON	38	33	47	0.5352	0.4471	0.5149
M:PUNCT	3	7	19	0.3	0.1364	0.2419
M:VERB	15	21	53	0.4167	0.2206	0.3538
M:VERB:FORM	6	2	4	0.75	0.6	0.7143
M:VERB:TENSE	14	14	49	0.5	0.2222	0.4
R:ADJ	22	25	79	0.4681	0.2178	0.3806
R:ADJ:FORM	3	0	2	1.0	0.6	0.8824
R:ADV	6	13	67	0.3158	0.0822	0.2013
R:CONJ	0	2	9	0.0	0.0	0.0
R:CONTR	1	0	0	1.0	1.0	1.0
R:DET	63	48	123	0.5676	0.3387	0.5
R:MORPH	60	35	44	0.6316	0.5769	0.6198
R:NOUN	195	168	512	0.5372	0.2758	0.4516
R:NOUN:INFL	16	1	0	0.9412	1.0	0.9524
R:NOUN:NUM	71	31	48	0.6961	0.5966	0.6736
R:NOUN:POSS	1	0	0	1.0	1.0	1.0
R:ORTH	100	49	75	0.6711	0.5714	0.6485
R:OTHER	199	248	768	0.4452	0.2058	0.3612
R:PART	9	13	22	0.4091	0.2903	0.3782
R:PREP	175	145	241	0.5469	0.4207	0.5159
R:PRON	41	24	69	0.6308	0.3727	0.5541
R:PUNCT	7	7	11	0.5	0.3889	0.473
R:SPELL	364	74	152	0.8311	0.7054	0.8025
R:VERB	82	103	345	0.4432	0.192	0.3513
R:VERB:FORM	87	55	70	0.6127	0.5541	0.6
R:VERB:INFL	7	1	0	0.875	1.0	0.8974
R:VERB:SVA	35	34	34	0.5072	0.5072	0.5072
R:VERB:TENSE	75	60	206	0.5556	0.2669	0.4568
R:WO	27	14	56	0.6585	0.3253	0.5466
U:ADJ	4	8	16	0.3333	0.2	0.2941
U:ADV	5	20	38	0.2	0.1163	0.1748
U:CONJ	0	1	5	0.0	0.0	0.0
U:CONTR	2	1	2	0.6667	0.5	0.625
U:DET	65	48	92	0.5752	0.414	0.5337
U:NOUN	7	8	45	0.4667	0.1346	0.3125
U:OTHER	9	49	77	0.1552	0.1047	0.1415
U:PART	2	0	8	1.0	0.2	0.5556
U:PREP	48	42	85	0.5333	0.3609	0.4868
U:PRON	9	3	30	0.75	0.2308	0.5172
U:PUNCT	1	3	16	0.25	0.0588	0.1515
U:VERB	4	8	31	0.3333	0.1143	0.241
U:VERB:FORM	3	4	3	0.4286	0.5	0.4412
U:VERB:TENSE	15	6	29	0.7143	0.3409	0.5859

Figure 8.3: The full ERRANT fine-grained, error-wise, evaluation from round 5. Model: `byt5`