Computational Linguistics in the Netherlands Journal 13 (2024) 147-171 Submitted 02/2024; Published 03/2024

Historical Dutch Spelling Normalization with Pretrained Language Models

Andre Wolters[*]
Andreas van Cranenburgh[**] A.W.VAN.CRANENBURGH@RUG.NL

[*] *Independent Scholar, Groningen, The Netherlands*

[**] *Department of Computational Linguistics, University of Groningen, The Netherlands*

Abstract

The Dutch language has undergone several spelling reforms since the 19th century. Normalizing 19th-century Dutch spelling to its modern equivalent can increase performance on various NLP tasks, such as machine translation or entity tagging. Van Cranenburgh and van Noord (2022) presented a rule-based system to normalize historical Dutch texts to their modern equivalent, but building and extending such a system requires careful engineering to ensure good coverage while not introducing incorrect normalizations. Recently, pretrained language models have become state-of-the-art for most NLP tasks. In this paper, we combine these approaches by building sequence-to-sequence language models trained on automatically corrected texts from the rule-based system (i.e., silver data). We experiment with several types of language models and approaches. First, we finetune two T5 models: Flan-T5 (Chung et al., 2022), an instruction-finetuned multilingual version of the original T5, and ByT5 (Xue et al., 2022), a token-free model which operates directly on the raw text and characters. Second, we pretrain ByT5 with the pretraining data used for BERTje (de Vries et al., 2019) and finetune this model afterward. For evaluation, we use three manually-corrected novels from the same source and compare all trained models with the original rule-based system used to generate the training data. This allows for a direct comparison between the rule-based and pretrained language models to analyze which yields the best performance. Our pretrained ByT5 model finetuned with our largest finetuning dataset achieved the best results on all three novels. This model not only outperformed the rule-based system, but also also made generalizations beyond the training data. In addition to an intrinsic evaluation of the spelling normalization itself, we also perform an extrinsic evaluation on downstream tasks, namely parsing and coreference. Results show that the neural system tends to outperform the rule-based method, although the differences are small. All code, data, and models used in this paper are available at https://github.com/andreasvc/neuralspellnorm

1. Introduction

Nineteenth-century Dutch spelling differs from modern spelling in certain aspects. Many words used to be written with double vowels, e.g., *hooren/horen*, *loopen/lopen*, and *oogen/ogen*. Some words ending with *-sch* are now reduced to *-s*, such as *mensch/mens* and *Nederlandsch/Nederlands*. The final *-n* case marking is no longer used, e.g., *dien/die* or *iederen/iedere*. Consider the following fragment of *Max Havelaar* by Multatuli taken from the OpenBoek corpus (van Cranenburgh and van Noord, 2022):

(1) Wat drommel kon [*@alt die dien*] [*@alt oude ouden*] heer bewegen , zich uittegeven voor een aanbidder van mijn zusje Truitje die [*@alt zere zeere*] [*@alt ogen oogen*] had , of van mijn [*@alt broer broêr*] Gerrit die altijd met zijn neus speelde?

Here the differences between 19th-century and modern Dutch spelling are highlighted in brackets. While these differences will not render a text unreadable for a human reader, the performance of NLP systems not trained on or adapted for the old spelling will degrade.

One approach for dealing with spelling variation is to apply a rule-based system to convert these old texts to modern spelling. However, building such systems requires careful engineering and a deep understanding of Dutch grammar and spelling rules. Moreover, a rule-based system is limited to a predefined set of corrections it can make, and expanding coverage without introducing incorrect changes is a challenge.

With the recent advancements made in the field of natural language processing (NLP), pretrained language models have become state-of-the-art for many NLP tasks, such as machine translation, text classification, or entity tagging. Such models require a large amount of training data. While human annotation takes a considerable amount of time and resources, we can also utilize a rule-based system to generate 'silver' data for training. Therefore, leveraging the power of language models combined with silver data from a rule-based system, we may be able to get improved spelling normalization performance without having to do any additional annotation. Since various language model architectures exist, ranging from word-based to character-based models, we can experiment with which approach works best. By developing language model capable of automatically normalizing 19th-century Dutch text, various NLP tasks may benefit from the normalized text, resulting in improved performance overall.

1.1 Research questions

In this study, we will answer the following research questions:

1. Which type of language model yields the best performance for the task of historical Dutch spelling normalization?

2. To what extent do the pretrained language models outperform the rule-based system in terms of spelling normalization performance?

3. Is a pretrained language model finetuned with silver data derived from a rule-based system able to make generalizations not found in the training data?

4. What is the effect on downstream tasks when normalizing spelling with pretrained language models, compared to other spelling normalization methods?

1.2 Outline

This paper is structured as follows: In Section 2, we discuss related work on the topic of spelling normalization and discuss reforms the Dutch spelling underwent in the 19th century. In Section 3, we discuss the training data for finetuning and pretraining our models, the gold data to evaluate our models, and the pre-processing steps applied to the data beforehand. In Section 4, we discuss our two experiments and discuss our evaluation metrics and strategy. In Section 5, we report and discuss our results, do an error analysis on the actual mistakes, present an extended analysis, and discuss a combination of both systems; in addition, we evaluate our systems on downstream tasks. In Section 6, we answer our research questions, discuss limitations, and discuss possible future work. Lastly, the annotation guidelines can be found in Appendix A.

2. Background

This section is divided into two parts. In the first part, we will discuss the major reforms to the 19th-century Dutch spelling, based on the overview by Nunn (2006, appendix H). In the second part, we will look into different approaches for the task of spelling normalization, ranging from different languages and model architectures to spelling correction and OCR post-correction.

2.1 19th century Dutch spelling reforms

Before the 19th century, there was no standardized Dutch spelling and there were several cases on which there was no agreement. In 1801 the Dutch government appointed Matthijs Siegenbeek (professor in linguistics at Leiden University) to devise a uniform spelling. Siegenbeek thought the Dutch spelling should be similar to the Dutch pronunciation; therefore, he made changes based on the principles of uniformity, etymology, and analogy. Word uniformity means that words of the same kind should also be written similarly. Word etymology, or word origin, examines how a word got its meaning and developed throughout history. Lastly, word analogies look at how two words are alike by comparing their different aspects. Based on these considerations, the spelling *gebrekig* was changed into *gebrekkig* (flawed), since he argued that the word was already written with double *k* in the plural form, and the word *gesprekken* (conversations) was also written with double *k*. Other changes were the introduction of the *lange ij* (long y), so words like *yzer* became *ijzer* (iron). In 1804 the government made Siegenbeek's spelling official, but it did not become popular. Writers and poets disagreed with the spelling reforms, such as Willem Bilderdijk; he was against the new spelling. Moreover, he published some of his own changes, for instance: *pligt* became *plicht* (duty) or *gooijen* became *gooien* (to throw). Bilderdijk's spelling was popular from 1830 to 1840 among some writers and poets.

In 1851, a conference was held named the *Taal- en Letterkundig Congres* (Linguistic and Literary Congress) in Brussels. The Netherlands and Belgian were tasked with creating a comprehensive dictionary named: (WNT; Dictionary of the Dutch Language) in which different vocabularies of the past century were published. There was only one problem, which spelling variant to use? There were two variants: the Siegenbeek and the poet Bilderdijk spelling. To solve this problem, the linguists Matthias de Vries and L.A. te Winkel merged the two variants into one uniform spelling, later named the *De Vries and Te Winkel* spelling. They agreed with Siegenbeek that the spelling shouldn't differ much from the pronunciation, meaning they kept numerous rules from Siegenbeek. Words that did change from the Siegenbeek variant were words with a *g*-sound, such as *kaghel* (heater) or *lagchen* (laugh). These became *kachel* and *lachen* since they argued it suited better with the pronunciation of the words. Moreover, the *g*-sound was now written in the same way as for the names: *Jochem* or *Lochem*. Lastly, the De Vries and Te Winkel spelling introduced rules for word hyphenation, for instance: the word *koningen* (kings), should it be split in *koning-en* or *konin-gen*? Other rules were about writing specific words as one or as two separate words. In 1864 the Belgian government accepted the new spelling, six years later the Dutch government followed, making the *De Vries and Te Winkel* spelling the official norm.

In 1954, the Dutch spelling was reformed again, and an official word list was published as *Het Groene Boekje* (the green book). Words ending with *-sch* were simplified to *-s*, double vowels (*oo*, *ee*) were simplified to single vowels (*o*, *e*), and the case endings *-n* became optional.

The most recent spelling reform was in 1995. The major change in this reform concerned the *-n* between noun compounds. Before, this was not consistently applied; for example, *kippenhok* (chicken coop) but *kippesoep* (chicken soup). In the new spelling, the -n should generally be used (although there are exceptions). In addition, variant spellings such as *aktie* instead of *actie* were abolished.

2.2 Spelling normalization

Spelling normalization is the task of converting words with various spelling forms into one uniform variant, ranging from misspelled to historical spelling. In natural language processing (NLP) it can be an important pre-processing technique since it reduces the text's out-of-vocabulary (OOV) words. This can increase the accuracy of other tasks in NLP, such as text classification, machine translation, or sentiment analysis.

2.2.1 Rule-based models

Van Cranenburgh and van Noord (2022) presented a rule-based system built for converting 19th-century Dutch spelling to its modern equivalent. The system was designed to only correct the spelling of the words, without altering sentence structure. For the construction of the rules, they applied two techniques. For the first technique they used:

- The DBNL novels corpus (van Cranenburgh et al., 2022), which contains over 130 million tokens from 1346 Dutch novels from the 18th to 20th century
- The Alpino parser (van Noord, 2006), which is a dependency parser for Dutch and has a lexicon of 200k words and 350k names
- Rule templates, which contain spelling changes for the 19th century. These are changes such as *-sch* → *-s*.

To construct the spelling rules, they applied the following steps:

1. Parse each novel with Alpino. If Alpino does not recognize a word, check if it meets the frequency threshold (equal to or greater than 10 in the corpus).
2. Apply the rule templates on the word to check if it can be altered; if so, check if Alpino recognizes the altered word; if successful, a new spelling rule is added.

This process resulted in a list of roughly 4000 words with old and modern spellings, such as *aandeelen* → *aandelen* (stocks). For the second technique, they manually inspected the novel *Eline Vere* by Louis Couperus to create an additional 432 rules, as well as context-sensitive rules expressed as regular expression substitutions (`sed`). The system thus consists of a mix of automatically and manually constructed rules. The rules are designed to favor precision over recall since the corrections made had to have a minimum frequency of 10 instances.

2.2.2 BERT models

Van 't Hof et al. (2022) consider the task of post-correction of optical character recognition (OCR) for historical Dutch text from the seventeenth up to the twentieth century; i.e., removing mistakes made by the OCR software. This differentiates from spelling normalization since OCR post-correction can include missing words or just wrong words. Moreover, the mistakes are less consistent; i.e., the same word can appear in multiple variants. With spelling normalization, we expect the spelling of words to be consistent throughout the text (e.g., *menschen*, humans). Still, OCR post-correction includes spelling correction, so the techniques used can apply to spelling normalization.

Van 't Hof et al. (2022) compared three models: dictionary, Word2vec (Mikolov et al., 2013), and BERTje (de Vries et al., 2019) for three different tasks: error detection, error correction, and a combination of both. The data comprised Dutch text (newspapers, literature, books, and radio bulletins). The task involved detecting and predicting the missing or misspelled word. Word2Vec and BERTje are used to predict missing words based on patterns of word associations in each sentence. BERTje produces contextual word embeddings that consider the context of the other words in the sentence. Hence, it generates a unique representation of each word based on the words around it in the sentence. In contrast, Word2Vec uses a static embedding layer where the representation of the word is the same for each sentence. BERTje was pretrained for the MLM (masked language modeling) objective in a self-supervised way, meaning random tokens were masked, and based on the remaining sentence, it has to predict the missing word. For instance:

- De [MASK] waren hard aan het werk. (The [MASK] worked hard.)

Based on the remaining context, BERTje has to predict the missing word, which should be *mensen* (human) in this example. This process was repeated for millions of sentences for multiple epochs. In the end, BERTje did not outperform Word2Vec for the historical Dutch text with OCR errors,

scoring several accuracy and F1 points lower on both error detection and error correction. The reason BERTje performed worse than Word2vec can be linked to two possible reasons.

First, the historical data comprised multiple periods (16th–19th century), resulting in certain words having multiple variants because spelling reforms for the Dutch language only became standardized at the beginning of the 19th century, making it more challenging for BERTje to correctly identify and correct the OCR mistakes. Second, BERTje was used without first finetuning it on the OCR correction task. BERTje's pretraining data did contain Dutch historical novels, but finetuning could have yielded improved performance.

Threshold	Precision	Recall
0.25	0.699	0.701
0.50	0.787	0.618
0.75	0.853	0.506
0.95	0.916	0.234

Table 1: Results for the RoBERTa-large token model for error detection on OCR generated texts (Table taken from Kim et al. 2021)

Kim et al. (2021) also focused on OCR error detection. Their study used a token-based BERT model, RoBERTa-large (Liu et al., 2019) with a token classification head. The data came from the HathiTrust corpus, which contains over 96k books. They only selected books for which there were multiple versions, such they can be compared to each other for any differences (including OCR mistakes). After selecting these duplicates, they applied sentence alignment, meaning that misspelled or missing words were aligned for the sentence pair. This resulted in sentence pairs of which one was used as ground truth and the other had the OCR error(s).

After sentence alignment, they used as input the OCR error and as label the ground truth from each sentence pair. The last step before training RoBERTa was tokenizing the input and labels. RoBERTa can tokenize certain words into sub-tokens, meaning that words not in RoBERTa's vocabulary are split into sub-tokens. Therefore, sub-tokens from both the input and labels were aligned, and only OCR error tokens were given an attention value of one. Table 1 shows precision and recall scores at different thresholds on the test set. At a threshold value of 0.95, the precision was 91.6%, which the researchers argued is more important than the recall since one would like to make confident corrections to the text instead of catching all errors. This was also the case for the rule-based system, favoring confident corrections (precision) over catching all errors (recall).

2.2.3 Seq2Seq models

Soper et al. (2021) focused on OCR post-correction of historical newspapers, which are prone to spelling errors and missing words. Instead of focusing on a BERT-based approach (encoder model), they view the problem as a translation task and used a sequence-to-sequence (encoder+decoder) model named BART (Lewis et al., 2020). BART combines a bi-directional encoder with an auto-regressive decoder layer, meaning BART can process and generate new sentences, unlike BERT, which is limited to generating only one token (label) per input sequence. The data came from the ICDAR 2017 Post-OCR Correction Dataset (Chiron et al., 2017), which consists of French and English historical newspapers. Since BART is pretrained for English, they used English newspapers only, resulting in a training set of 38,975 sentences. OCR post-correction consists of different corrections made to the text. The five OCR error types BART was trained for were:

- Over-segmentation, which means words written separately should be merged into one (e.g., in decent → indecent)

- Under-segmentation, which is the opposite of the latter described, splitting words that are written as one (e.g. andjust → and just)
- Misrecognized characters, which are words where one or more characters should be corrected (e.g. ipto → into)
- Missing characters, which are words where one or more characters are missing (e.g. hat → what or iterestng → interesting)
- Hallucination, which consists of words or strings that should be deleted altogether.

It should be noted that these errors can also appear together in the text, for instance: words containing both under-segmentation and missing characters. In the end, BART detected and corrected all five OCR error types, improving the text accuracy by 29.4%. BART was less consistent with correcting under-segmentation errors due to the low frequency of these errors in the training dataset.

Kim et al. (2021) also implemented a seq2seq approach, namely T5-base, where the input (source) was the OCR error sentence, and the output (target) was the corrected sentence. They trained the base version for only three epochs but added a special `<OCR>` tag to denote the beginning and end of an OCR error in the sentence. In the end, the model corrected six times as many errors as it introduced, which meant it did introduce errors previously not in the text. The authors did mention that they used a seq2seq model with a sub-word tokenizer, meaning the model encodes words into sub-words. In contrast, they argued that a character-based model would be a better fit, but this would be more computationally costly.

3. Data

This section is divided into three parts; first, we will discuss the training corpora used for our two experiments, namely, a finetuning and a pretraining dataset. Secondly, we discuss the gold data used to validate and test our trained models. Lastly, we will describe the pre-processing and cleaning steps applied to the data before using it with our models.

3.1 Training data

Since our task involved finetuning for a downstream task, we needed data to train our models. To the best of our knowledge, no large parallel corpora were available for 19th-century Dutch spelling normalization; therefore, we collected our own. For this, we used Project Gutenberg, which offers over 60k public domain electronic texts, from which over 1k are 19th century Dutch texts. We selected books published between 1840–1917; this selection was made since the Dutch language underwent numerous reforms as explained in Subsection 2.1, therefore picking from each period allowed for a diverse set of spelling variations. We preprocessed the texts from Project Gutenberg using the Dutch Literature Pipeline.[1] This tool allowed to easily scrape texts from Project Gutenberg by the number of tokens and clean and convert the text into a standard format accepted by the rule-based system. An example of how this tool returned a selected part from a novel can be seen in Figure 1.

Here, the small paragraph gets split up into two separate sentences, each with an index value where the first number refers to the paragraph and the second to the sentence in that paragraph. Each sentence is tokenized by splitting words and punctuation symbols.

3.1.1 Creating silver data

After collecting the data, we were left with pre-processed 19th-century Dutch sentences split into paragraphs. We opted to go with an automatic annotation strategy since annotation by hand takes considerably more time and resources, and the rule-based system by van Cranenburgh and van Noord (2022) was already shown to be quite effective at normalizing 19th-century Dutch spelling.

1. https://github.com/andreasvc/dutchlitpreproc

Input	Zij gingen samen een paar passen voort; toen haalde Frank den sleutel uit zijn zak—den sleutel van White-Rose. Hij opende de deur; een zeshoekige Moorsche lantaren scheen in de vestibule zacht met halve vlam.
Output	26-1|Zij gingen samen een paar passen voort ; toen haalde Frank den sleutel uit zijn zak - den sleutel van White-Rose . 26-2|Hij opende de deur ; een zeshoekige Moorsche lantaren scheen in de vestibule zacht met halve vlam .

Figure 1: An example of the input and output of the Dutch Literature Pipeline tool used to scrape data from Project Gutenberg (example derived from novel Noodlot by Louis Couperus)

Novel	Author	Published	Tokens
1. Noodlot	Louis Couperus	1890	8022
2. Camera Obscura	Nicolaas Beets	1839	8012
3. Waarheid En Droomen	Johanas Hasebroek	1840	8003
4. De kleine Johannes	Frederik van Eeden	1885	8022
5. Naar het middelpunt der Aarde	Jules Verne	1864	8002
6. De stille kracht	Louis Couperus	1900	8021
7. Ferdinand Huyck	Jacob van Lennep	1840	8026
8. Een verlaten post	Johanna van Woude	1891	8005
9. Slechte Tijden	Charles Dickens	1854	8002
10. De lelie van 's-Gravenhage	Jacob Cremer	1878	8002
11. Vogels van diverse pluimage	Carel Vosmaer	1872	8009
Total Tokens 5k			88.126
12. In Griekenland De Aarde en haar Volken	Anastase Adossidès	1909	2458
13. Instituut Sparrenheide	Chris van Abkoude	1917	8002
14. De Ellendigen	Victor Hugo	1862	8003
15. De Geschiedenis van Woutertje Pieterse	Multatuli	1890	8010
16. Uit het leven van Dik Trom	Cornelis Johannes Kieviet	1891	8016
17. Zes maanden bij de commando's	Nico J. Hofmeyr	1903	8002
18. De Pleegzoon	Jacob van Lennep	1833	8066
19. Indische Menschen in Holland	Maurits	1890	8003
20. Een Twaalftal Samenspraken	Desiderius Erasmus	1906	8016
21. Indische Huwelijken	Annie Foore	1895	8019
Total Tokens 10k			162.721

Table 2: Literature used to construct the training data (1-11 for train set 5k, and 1-21 for train set 10k), tokens refer to the number of words, including punctuation symbols.

The system works by iterating through a sentence at word level and checking for matches in the automatic and manual rules; if there is a match, the word is enriched with a meta-annotation that describeds the normalized version, as shown in Figure 2. In the example, the words *oogenblik* (moment/instant) and *eenig* (only/sole) are normalized. The newly created words between brackets start with an Alpino meta annotation tag; `@alt` is the most common form, where the old word (left one) is to be replaced with the modern one (right side). This format ensures that everything remains aligned with the original text. Other meta annotation tags are:

- `@alt_mwu`: Used when two or more words in the original should be merged merged into one, and there is a different spelling, e.g., `[ @mwu_alt zoiets zoo -iets ]`

Input	47-3|Voor het oogenblik schenen die handen zijn eenig verdriet te zijn .
Output	47-3|Voor het [@alt ogenblik oogenblik] schenen die handen zijn [@alt enig eenig] verdriet te zijn .

Figure 2: An example of the expected input and output of the rule-based system (example derived from novel Noodlot by Louis Couperus)

- `@mwu`: Used when two or more words in the original should be merged into one (but without a change in spelling), e.g., `[ @mwu_alt van daag ]`
- `@phantom`: Used when a word in the original should be split into multiple tokens, e.g., `[ @phantom op ] [ @phantom te ] [ @alt merken optemerken ]`

Table 2 shows the list of novels use as training data. We used two different-sized finetuning datasets to investigate the effect of increasing the amount of training data. We selected the first 11 novels for constructing our 5k training dataset, which had 88k total tokens, and all novels for the 10k dataset with 163k tokens. From each novel, we selected the first 8000 tokens, rounded to the next sentence boundary.

3.2 Pretraining data

Our second experiment involved pretraining, so we needed a large corpus of Dutch text. We used the pretraining corpus from BERTje (de Vries et al., 2019). BERTje is a monolingual Dutch version of the standard BERT model (Devlin et al., 2019). It is pretrained on roughly 12GB of Dutch text, consisting of five sources (books, newspapers, SoNaR, web news, and Wikipedia). It performs for downstream tasks such as text classification and entity tagging better than the multilingual BERT model for the Dutch language. We selected only the books and SoNaR corpora since the books consist of modern and historical novels, the same type as our source and target in the training data. The SoNaR corpus (Schuurman et al., 2010) is a reference corpus of standard written Dutch which is the same type as our target sentences in the training data. The Books corpus contained a total of 7054 novels with a total of 60 million sentences. To limit the amount of time required for pretraining, we selected the first 500 sentences with at least 50 characters from each novel. This leaves us with 2.25 million sentences, from which we selected the first 2 million as training and the last 200k as validation data. The SoNaR corpus contained 31 million sentences, split out over 20 smaller text files; therefore, we selected all sentences which were longer than 50 characters, which left us with 18 million sentences. From these remaining sentences, we again select the first 2 million as training and the last 200k as validation data.

3.3 Gold data

For the gold data, the OpenBoek repository from van Cranenburgh and van Noord (2022) already contained three manually corrected novel fragments. We used these novels to evaluate our trained models and the rule-based model by van Cranenburgh and van Noord (2022). The human corrections have been done by three different annotators; therefore, there are some inconsistencies between them. To overcome this issue, we updated the annotation guidelines to clear up some ambiguities. The most important changes in the guidelines are the following cases:

- Leave the hyphenation in hyphenated words unchanged, (e.g. twee-en-negentig, op-zichzelf), but do correct their spelling where needed (e.g., Noord- Duitschland → Noord-Duitsland, kampioen-athleet → kampioen-atleet)
- Keep archaic verb conjugation of words if the spelling is correct, e.g., *hij zeide*, *gij kwaamt*.
- Remove circumflex accents; e.g., *elkaâr* or *broêr*.

Novel	Author	Published	Tokens
Max Havelaar	Multatuli	1860	9977
Sherlock Holmes: De Agra-Schat	Conan Doyle	1899	9986
Titaantjes	Nescio	1915	11786
Total Tokens Test			31.749
De Uitvreter	Nescio	1911	14298
Anna Karenina	Lev Tolstoj	1877	10089
Total Tokens Validation			24.387

Table 3: Literature used for the gold data (dev/test data), tokens refer to the number of words, including punctuation symbols split on whitespace.

- Correct words in French spelling to Dutch alternatives, if possible For example: *billardballen* → *biljartballen*; *million* → *miljoen*.

Additional changes were examples already in the guidelines but overlooked by the former annotators. The complete revised annotation guidelines are in Appendix A.

We applied the updated guidelines to the existing novel fragments to ensure consistency. We also corrected two extra novels, which were first automatically corrected with the rule-based system; these novels are used as validation data during the training of our models. The novels used as gold data are listed in Table 3.

3.4 Data pre-processing

For our training dataset, we used a total of 21 novels. Since the data was automatically annotated, some sentences were corrupted and omitted from the final datasets. These sentences contained only punctuation such as empty parentheses, which is not useful as training data. Some sentences also had corrupted characters. Instead of correcting these sentences, which meant manual correction, we decided to leave them out altogether to ensure the data was purely silver standard. We did remove the front matter for each novel, such as titles, authors, and year of publication. In total, this was less than 0.1% of the data. The pretraining datasets were obtained from BERTje and had already undergone pre-processing and cleaning. Similarly, the gold data was also already cleaned and corrected.

4. Method

This section will describe the two experiments run with the datasets, namely, (1) Finetuning FlanT5 and ByT5 with our silver data, and (2) Pretraining ByT5 with monolingual Dutch data and finetuning it afterward with the same datasets from experiment 1. We will discuss the model's architecture for each experiment, the environment we trained in, and the (hyper)-parameters we optimized. Lastly, we will discuss the evaluation metrics used for inference.

4.1 Experiment 1: finetuning

Our first experiment involved finetuning the two T5 models with the generated silver data as described in Section 2. We chose the T5 models since T5 comes in different configurations, and is one of the few that has both word and character based versions. We trained each model twice, first with only 5k sentences from our train dataset and secondly with our entire dataset of roughly 10k sentences, in order to assess the impact of the amount of traing data. The training was done on the RUG Hábrók

GPU cluster, which uses high-end GPUs (NVIDIA A100 or V100). For the finetuning we considered the following hyperparameters:

1. Learning rate: using values from 1e-4 to 5e-5, where higher learning rates performed better with 5k sentences and lower learning rates were better with the entire dataset.
2. Batch size: using the values 16 or 32; our models performed better when the data was fed in smaller batches.
3. Sequence length: here we distinguished between the train and validation set. For the train set, we used a maximum sequence length of 155 since sentences over 150 characters long are split beforehand. This is to use less computational resources in the training process for the larger model (ByT5) and speed up the overall training process. For the validation set, we used a maximum sequence length of 455 because the longest sentence in the development set was 450 characters long. Moreover, applying the same technique on the validation set is better because both models will predict on the test set without splitting.
4. Training epochs: we implemented an early stopper that monitored the validation set for the highest value accuracy. We used a patience of three, meaning if the *val_ accuracy* decreased three consecutive times, the model halted training and loaded the checkpoints with the highest accuracy. This ensured that overfitting was less likely, and our models could achieve their best setting.

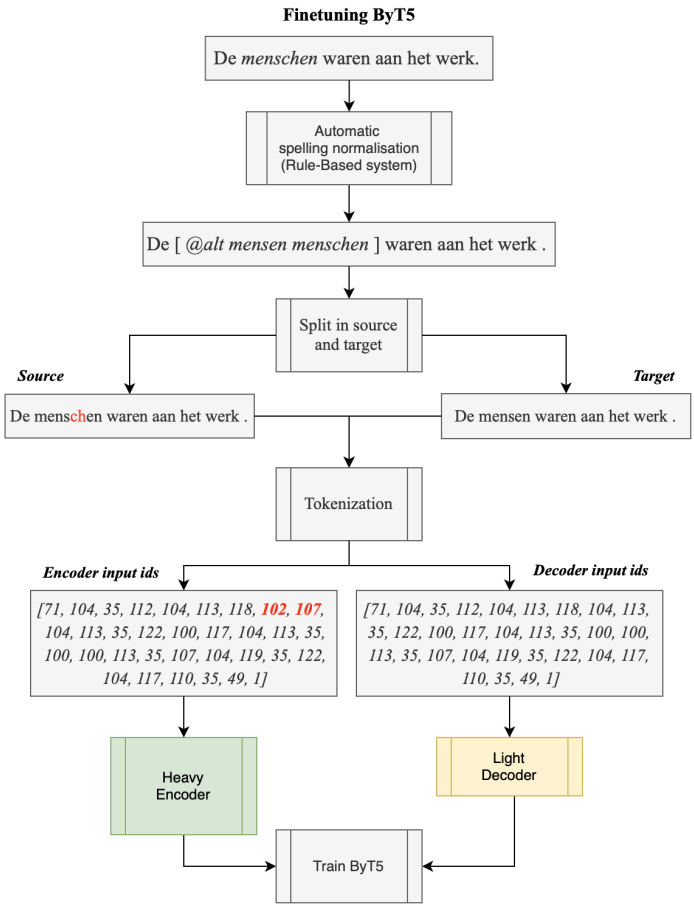

Figure 3: Overview of the finetuning process

In Figure 3, a schematic overview of the finetuning process is shown for our ByT5 model; the same approach is used for FlanT5, except that it uses a subword tokenizer instead of the character-based one. In the first step, the sentence is normalized with the rule-based system, where the input (source) is the 19th century spelling and the output (target) is the modern version. Afterward, we apply the tokenization process, which involves converting the text to numerical data, so our model can work with it. Since this example uses the token-free model, each character from the sentence gets tokenized, and since our source sentence is longer than the target, two extra characters and input ids are present, highlighted in red.

4.1.1 FlanT5

The first model we used was Flan-T5 small, which has 77 million trainable parameters. We chose this variant of T5 since we work on Dutch, and the original T5 is only pretrained on English (monolingual). Moreover, since this version is nearly the same size as the original T5 (60.5 million parameters), it is less computationally heavy than the multilingual version of T5, mT5 (Xue et al., 2021), which has over 300 million trainable parameters. The only difference between Flan-T5 and mT5, apart from their architecture and total parameters, is the number of supported languages. Flan-T5 supports up to 60 languages, while mT5 supports 101 languages. This is because mT5 is pretrained on the multilingual C4 corpus (mC4), while Flan-T5 is not pretrained but further finetuned for 1.8k tasks from a newer T5 checkpoint, which is only pretrained on the C4 corpus (English only). This means we use a further finetuned version of T5 instead of a completely new pretrained model.

Architecture Flan-T5 is based on the newer released checkpoint of T5: T5 version 1.1, which has 77 million parameters. Version 1.1 is trained with the same C4 dataset, only this time, the rectified-linear (ReLu) activation function is replaced with a variant of gated-linear-units (GLU) named GEGLU (Shazeer, 2020). Version 1.1 is also pretrained in an unsupervised way; therefore, it cannot be used on a downstream task but must be finetuned first. The authors of Flan-T5 finetuned T5-v1.1 with 473 datasets for 146 different task categories, resulting in 1836 total tasks. They applied instruction finetuning, which means the model gets finetuned for multiple downstream tasks by applying a special prefix.

Text	mT5 input_ids	length	ByT5 input_ids	length
John	[4040]	1	[77, 114, 107, 113]	4
is	[339]	1	[108, 118]	2
walking	[259, 42822]	2	[122, 100, 111, 110, 108, 113, 106]	7
his	[1638]	1	[107, 108, 118]	3
German	[20567]	1	[74, 104, 117, 112, 100, 113]	6
Shepherd	[320, 125326]	2	[86, 107, 104, 115, 107, 104, 117, 103]	8
.	[259, 260]	2	[49]	1
Total seq:		10		31
Vocab size:	250100		384	

Table 4: An example of the encoding process (input_ids) for a sub-word model (mT5) and token-free model (ByT5), special tokens are excluded from the example.

4.1.2 ByT5

The second model we trained was ByT5 small, which has 300 million trainable parameters, comparable to mT5 small. We chose this model because a character-based model seems like a good fit for the task of spelling normalization. ByT5 does not use a tokenizer but operates directly on the raw characters;

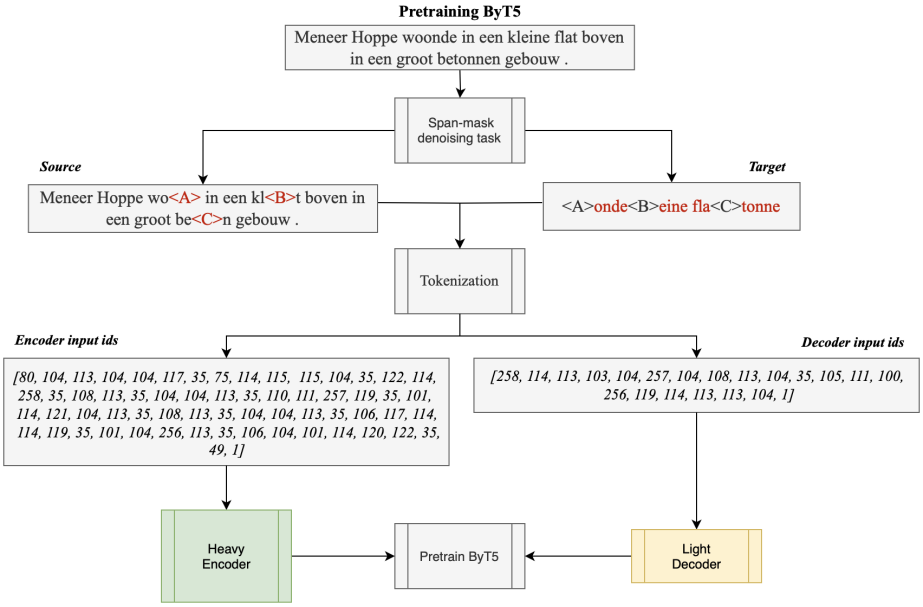

Figure 4: Overview of the pretraining method

therefore, the model is not restricted to a single language since the vocabulary is not subwords but UTF-8 bytes. Moreover, spelling normalization focuses on characters; we have to make corrections on the character instead of word level. Furthermore, since ByT5 encodes each character of the sentence separately, it may be able to generalize between spelling mistakes across different words.

Architecture ByT5 is based on the mT5 architecture, a multilingual version of T5. The major different with mT5 is the lack of a tokenizer. While mT5 uses the SentencePiece tokenizer (Kudo and Richardson, 2018), ByT5 is fed with bytes (characters) as input directly. This means that ByT5 encoded sequences are longer than mT5 since each word is converted into multiple character tokens depending on the number of characters, whereas with a subword tokenizer, it depends if the word is in the tokenizer's vocabulary. An example of the tokenization process between mT5 and ByT5 can be found in Table 4; here, both models have advantages. mT5 has a shorter sequence length, which means the total computational cost is lower, but ByT5 has a smaller tokenizer vocabulary, which means the chance of having out-of-vocabulary words (<UNK>) is less likely to occur.

Another difference from mT5 is the distribution between the encoder and decoder layers. mT5 has 16 layers evenly distributed between the encoder and decoder, while ByT5 has an encoder with 12 layers and a decoder with just 4 layers. This means that ByT5 relies more on its encoder. Lastly, regarding the size of the vocabulary, since ByT5-small only uses 384 unique characters in its tokenizer, the overall percentage of the model's size dedicated to the vocabulary is only 0.3%, while mT5-small vocabulary takes up 85% of the total size.

4.2 Experiment 2: pretraining and finetuning

In our second experiment, instead of only finetuning, we start by pretraining our best model from experiment 1 (ByT5) and afterward finetune with the same silver data as in experiment 1. Since pretraining takes considerably more computational power, we limited our pretraining datasets to only 2 million sentences, as discussed in Subsection 3.2. For pretraining, we made use of the Flax script from HuggingFace,[2] supplied by the authors of T5. Flax is a machine learning library from

2. https://github.com/huggingface/transformers/blob/main/examples/flax/language-modeling/run_t5_mlm_flax.py

Google that outperforms TensorFlow and PyTorch in performance and makes pretraining more computationally efficient. We again used the Hábrók GPU cluster for pretraining and ran on an A100 NVIDIA GPU for roughly 50 hours. The main difference between pretraining over finetuning is that pretraining is done self-supervised, meaning ByT5 is fed with unlabeled data from a large corpus. As mentioned previously, ByT5 has been pretrained with the mC4 corpus, which contains roughly 277GB of Dutch text. These texts are from the common crawl corpus, which contains petabytes of data scraped from the internet. Further pretraining on in-domain data can make the model more suited for the downstream task it will be finetuned for, and since our task involved spelling normalization for the Dutch language, letting the model pretrain on additional (modern) Dutch sentences could benefit its overall performance. Figure 4 shows a schematic overview of the pretraining task. Here, the main difference with our finetuning task is that instead of encoding two variants of a sentence (19th and modern Dutch) for the source and target, it is based on only one sentence. The pretraining task is a span-mask denoising task, which involves first corrupting up to 20 characters, which the ByT5 model then has to predict based on the remaining non-corrupted characters in the source sentence, as shown in the illustration.

4.2.1 Hyperparameters

For pretraining, we considered the following hyperparameters and values:

1. Learning rate: we used a value of 5e-4 in combination with a warm-up step of 10k and weight decay of 0.001.
2. Batch size: we used a value of 8 per device, meaning for each GPU/CPU.
3. Maximum sequence length: we used a value of 512; longer sentences are truncated
4. Span length of masked tokens: we applied a value of 20, meaning that up to 20 characters may be corrupted at a time.

We pretrained the models for 25 epochs, saved the weights after every 50k training steps, and evaluated the validation set after every 10k training steps.

4.3 Spelling normalization evaluation

For the evaluation of the trained systems, we focused on the following four intrinsic metrics: precision and recall (Reynaert, 2008), Error Reduction Rate (ERR; van der Goot et al., 2021), and character n-gram F-score (ChrF and ChrF++; Popović, 2015, 2017). For the calculation of the precision and recall, we calculated it based on the whole text instead of a single sentence, meaning we computed the precision and recall per novel instead of averaging it per sentence. We used the `precision_score` and `recall_score` functions from the Scikit-learn library (Pedregosa et al., 2011).

Since our task focused on text normalization, the total number of tokens that had to be normalized per novel can differ. This means that an accuracy of 95% on one dataset can be quite good, while on another, it might be useless since we do not consider the total of corrections actually made to the text. Therefore, we also report ERR; this metric is the accuracy normalized by the number of words that must be corrected in the text. The formula for the ERR is as follows:

$$ERR = \frac{\%\text{accuracy} - \%\text{words\_not\_normed}}{100 - \%\text{words\_not\_normed}} \quad (1)$$

Here `%accuracy` is the percentage of correctly spelled words (standard accuracy), and `%words_not_normed` is the percentage of words that did not need to be normalized (baseline accuracy). Therefore, the ERR requires three versions of the test set: (1) the original, (2) the gold, and (3) the predictions. The baseline accuracy is calculated between the original and gold, and the standard accuracy is calculated between the gold and the prediction. We used the script from van der Goot et al. (2021) to calculate the ERR.

Lastly, we computed ChrF and ChrF++ scores for each sentence, which are based on character 6-grams. The formula for the ChrF is as follows:

$$\text{CHRF}_\beta = (1+\beta^2)\frac{\text{CHRP} \cdot \text{CHRR}}{\beta^2 \cdot \text{CHRP} + \text{CHRR}} \tag{2}$$

Here chrP stands for the character n-gram precision, and chrR for the character n-gram recall. The β value indicates the importance of recall, where a higher β value means more importance on the recall, and a β value of one means that precision and recall have the same weight for calculating the F-score. The ChrF++ score is a newer variant which not only considers character 6-grams, but also word bigrams. The scores for each sentence are averaged into a macro F-score.

It should be noted that for the computation of the latter described metrics, before calculating the metric, we first split the sentences into words and ensure that they are of equal length. Since both T5 models append the punctuation marks to the end of the word, we first pre-processed the data before evaluation. We also had differences in output length due to words being split or merged in the process of spelling normalization (e.g., *op te merken* vs. *optemerken*), resulting in longer or shorter sentences for the output. To overcome this issue, we applied the following pre-processing steps beforehand:

1. Split words and punctuation marks, but keep punctuation that should be part of the word (e.g., etc., IX.)
2. Align sentences by merging/separating words on white space that should be considered as one between the source and target (e.g., "op" "te" "merken" → "op te merken").

Prediction (11)	"En", "dan", "kon", "je", "ervan", "**opaan,**", "dat", "Bavink", "over", "Lien", "**begon.**"
Gold (14)	"En", "dan", "kon", "je", "ervan", "op", "aan", ",", "dat", "Bavink", "over", "Lien", "begon", "."
Prediction (13)	"En", "dan", "kon", "je", "ervan", "opaan", ",", "dat", "Bavink", "over", "Lien", "begon", "."
Gold (14)	"En", "dan", "kon", "je", "ervan", "**op**", "**aan**", ",", "dat", "Bavink", "over", "Lien", "begon", "."
Prediction (13)	"En", "dan", "kon", "je", "ervan", "opaan", ",", "dat", "Bavink", "over", "Lien", "begon", "."
Gold (13)	"En", "dan", "kon", "je", "ervan", "op aan", ",", "dat", "Bavink", "over", "Lien", "begon", "."

Figure 5: An example of a prediction and gold pair before and after applying pre-processing steps for evaluation. Upper row is beforehand, middle row is applying step 1, lower row is applying step 2, number in brackets refers to the sentence length (example derived from *Titaantjes* by Nescio)

An example of the pre-processing steps is shown in Figure 5. Here the final punctuation is part of the last word *begon* (began) for the prediction, and the words *op aan* (on) are seen as two words in the ground truth. First, the punctuation is split, and second the words *op aan* are joined together. Note that the whitespace remains, since for the evaluation *opaan* vs. *op aan* are treated as different tokens. If, after pre-processing, the prediction and ground truth are still unequal in length on word-level due to hallucinations of the models, we discard the prediction and use the original sentence (19th century) instead. This ensured that we could evaluate the whole test set, without skipping any sentence. Hallucinations were mostly sequences of words repeated multiple times from the input, resulting in longer sentences of up to 15 words. Since the rule-based output also made corrections where old spelled words were combined or separated, we applied the same pre-processing steps. The only difference was that the rule-based system didn't have any hallucinations.

5. Results & Discussion

In this section, we will report and discuss the results for our two experiments. We first discuss the results obtained by finetuning the initial two models for experiment 1 (Subsection 4.1). Second, we

report the results of our pretrained models from experiment 2 (Subsection 4.2). For both experiments, we compare the obtained results with the rule-based system by van Cranenburgh and van Noord (2022) and report the results on the four metrics mentioned in Subsection 4.3. Lastly, we discuss the error analysis for the actual spelling mistakes and do an extended analysis, diving deeper into specific predictions from the test set.

Novel	Model	ERR (%)		Precision (%)		Recall (%)	
		5k	10k	5k	10k	5k	10k
Max Havelaar	1. Rule-Based	70.63		96.02		96.20	
	2. FlanT5	55.44	61.52	94.19	94.85	94.76	95.21
	3. ByT5 (orig)	67.59	67.59	95.73	95.74	95.99	95.99
	4. pretrain ByT5 SoNaR	67.09	69.37	95.67	95.91	95.92	96.20
	5. pretrain ByT5 Books	67.09	**71.39**	96.03	**96.04**	96.28	**96.39**
Sherlock Holmes	1. Rule-Based	69.73		96.54		97.00	
	2. FlanT5	59.18	62.50	94.94	95.53	96.00	96.32
	3. ByT5 (orig)	72.07	72.27	96.93	96.93	97.58	97.59
	4. pretrain ByT5 SoNaR	72.27	72.27	96.97	96.89	97.52	97.56
	5. pretrain ByT5 Books	71.29	**74.02**	96.81	**97.28**	97.58	**97.87**
Titaantjes	1. Rule-Based	62.70		97.93		98.27	
	2. FlanT5	72.02	77.29	96.36	96.89	97.11	97.46
	3. ByT5 (orig)	82.11	83.03	97.69	97.81	98.13	98.31
	4. pretrain ByT5 SoNaR	83.94	**85.55**	98.08	98.21	98.30	**98.59**
	5. pretrain ByT5 Books	84.17	85.32	98.04	**98.26**	98.47	98.57

Table 5: ERR (error reduction rate), precision, recall results for 5k/10k trained models: 1. Rule-Based system by van Cranenburgh and van Noord (2022), 2. Flan-T5-small, and three ByT5-small models (3. ByT5 original weights, 4. pretrained ByT5 with the BERTje books corpus, 5. pretrained ByT5 with SoNaR).

5.1 Experiment 1

For experiment 1, we finetuned FlanT5 and ByT5, both from their original pretrained checkpoints. We finetuned each model first with the 5k silver data and later with the full 10k dataset. The results are in rows 2–3 of Table 5 and 6. In the following discussion we focus mainly on the ERR since it reflects only the tokens that need to be normalized.

For the 5k dataset, ByT5 outperformed FlanT5 on ERR, precision, and recall for all three novels, where the most significant difference can be seen for the novel *Sherlock Holmes*, scoring 21.8% higher on ERR. Precision and recall scores lie closer together, where precision scored 2.1% and recall 1.65% higher, but this is expected since it takes into account all the words in the novel. The same goes for the ChrF results; ByT5 performed better on all three novels for both ChrF scores.

For the 10k dataset, again, ByT5 scored the highest, where the most significant deviation for the ERR is again the novel *Sherlock Holmes*, scoring now only 15.6% higher. Finetuning FlanT5 with twice the data did benefit its performance for all three novels, whereas ByT5 had minimal increases, scoring for the novel Max Havelaar the same ERR and for the other two only a minimal increase of 0.3 & 1.10 percent, respectively.

Comparing the finetuned models with the rule-based model shows that FlanT5 only scored higher for the novel *Titaantjes*, both with the 5k and 10k dataset. ByT5 scored higher on 2 out of 3 novels (*Sherlock Holmes* and *Titaantjes*) when compared with the rule-based model.

Novel	Model	ChrF (%)		ChrF++ (%)	
		5k	10k	5k	10k
Max Havelaar	1. Rule-Based	98.86		98.68	
	2. FlanT5	98.43	98.58	98.14	98.34
	3. ByT5 (orig)	98.79	98.80	98.59	98.60
	4. pretrain ByT5 SoNaR	98.77	98.86	98.56	98.67
	5. pretrain ByT5 Books	98.80	**98.92**	98.60	**98.74**
Sherlock Holmes	1. Rule-Based	98.83		98.52	
	2. FlanT5	98.42	98.54	98.01	98.16
	3. ByT5 (orig)	98.87	98.89	98.60	98.62
	4. pretrain ByT5 SoNaR	98.85	98.84	98.58	98.58
	5. pretrain ByT5 Books	98.83	**98.92**	98.55	**98.67**
Titaantjes	1. Rule-Based	98.39		98.12	
	2. FlanT5	99.21	99.34	99.01	99.19
	3. ByT5 (orig)	99.45	99.47	99.34	99.36
	4. pretrain ByT5 SoNaR	99.26	99.54	99.23	99.45
	5. pretrain ByT5 Books	99.53	**99.56**	99.43	**99.46**

Table 6: ChrF and ChrF++ results for 5k/10k trained models: 1. Rule-Based system by van Cranenburgh and van Noord (2022), 2. Flan-T5-small, and three ByT5-small models (3. ByT5 original weights, 4. pretrained ByT5 with BERTje books corpus, 5. pretrained ByT5 with SoNaR).

5.2 Experiment 2

For experiment 2, we first pretrained ByT5 with two different datasets, the BERTje books corpus and SoNaR. Afterward, we finetuned both models in the same way as experiment 1. Results for the ERR, precision, recall, and ChrF(++) can be found in Table 5 and 6, rows 4–5, where again we mainly focus on ERR.

For the 5k dataset, for the novel *Max Havelaar*, both models have the same ERR score, but ByT5 pretrained on books achieves 0.38% higher on both precision and recall, and similarly for ChrF(++). For the novel *Sherlock Holmes*, pretrained SoNaR performed better on ERR (1.37% higher), and for *Titaantjes*, pretrained books performed better on ERR (0.27% higher). Resulting in pretrained books performing better for 2 out of 3 novels.

For the 10k dataset, the pretrained books version extends its lead, where its performance for the novel *Max Havelaar* on ERR is 2.9% higher over pretrained SoNaR. For *Sherlock Holmes*, pretrained SoNaR has a minimal increase, while pretrained books achieve 2.42% higher ERR over the pretrained SoNaR. For the last novel, *Titaantjes*, pretrained SoNaR has a more significant gain over books, scoring 0.27% higher on ERR. Therefore, finetuning with the bigger train set resulted in a larger deviation between the pretrained versions. Where pretrained books performed better overall for 2 out of 3 novels and had a minimal difference of only 0.27% in ERR.

Lastly, comparing the rule-based system with the two pretrained versions shows that both systems outperform it for the novels *Sherlock Holmes* and *Titaantjes*. Still, pretrained SoNaR didn't perform better for *Max Havelaar*. The only model capable of beating the rule-based system in all three novels was pretrained ByT5 books.

Novel	Rule-based top 10 mistakes			ByT5 top 10 mistakes		
	prediction	ground truth	freq	prediction	ground truth	freq
Max Havelaar	zo -iets	zoiets	5	meisjen	meisje	5
	op-eens	opeens	4	op-eens	op eens	4
	duitsch/duitsche	Duits	4	korrespondentie	correspondentie	3
	korrespondentie	correspondentie	3	optemerken	op te merken	3
	optemerken	op te merken	3	moeielijker	moeilijker	3
	moeielijker	moeilijker	3	luî	lui	2
	luî	lui	2	konnexie	connectie	2
	ouden	oude	2	duitsche	Duitse	2
	konnexie	connectie	2	enigen	enige	2
	enigen	enige	2	solieden	solide	2
Total mistakes:			116			113
Sherlock Holmes	zei	zeide	24	zei	zeide	24
	ene	een	13	ene	een	13
	enigen	enige	7	enige	enige	7
	uwe/uwen	uw	6	uwe/uwen	uw	6
	dezen	deze	4	dezen	deze	4
	gene	geen	3	bizonder	bijzonder	3
	haren	haar	3	oudsten	oudste	3
	onzen	onze	3	gene	geen	3
	te zamen	tezamen	2	onzen	onze	3
	zoëven	zo-even	2	zoëven	zo-even	2
Total mistakes:			155			133
Titaantjes	hij	-ie	93	der	van de	15
	der	van de	15	onzen	onze	2
	onzen	onze	2	verten	verte	2
	zooiets	zoiets	2	anderen	andere	2
	verten	verte	2	opaan	op aan	1
	anderen	andere	2	der	er	1
	opaan	op aan	1	effe	effen	1
	koeienoogen	koeienogen	1	geprakkizeerd	geprakkiseerd	1
	der	er	1	bizonder	bijzonder	1
	effe	effen	1	metdertijd	mettertijd	1
Total mistakes:			163			63

Table 7: Top 10 most common spelling mistakes for the rule-based system and our best-performing model of each novel. Total mistakes are the total number of spelling mistakes present in the novel for the system, e.g., for *Max Havelaar*; the rule-based had 116 incorrect and ByT5 only 113.

5.3 Error analysis

The previously described results only give us a limited view of our model's performance; therefore, we report the actual spelling mistakes the systems made on the word level, see Table 7. We compare the rule-based system with our best model for each novel's top 10 most common mistakes, including the total mistakes. The total mistakes are the sum of all wrong (precision) and missing (recall) normalizations. For the novel *Max Havelaar*, the top 10 is almost identical, meaning the difference has to be due to less frequent errors. The only difference is the first row, where *zoiets* (something like that) is done incorrectly by the rule-based, and *meisje* (girl) by the ByT5 model. For the first

case, the rule-based system does not correct the word, where it only corrected the first part: zoo → zo. For the second case, the word *meisje* does get corrected by the rule-based system, and since our model used it to generate the training data, it should have corrected the word. But due to the word not appearing in one of the 21 novels of the training data, our models never learned to correct it, the same problem Soper et al. (2021) described for their under-segmentation class. The difference for this novel is also the smallest, with only three fewer incorrect words, but this novel was also the oldest from the test set.

Looking at the novel *Sherlock Holmes*, the most frequent word is *zeide* (said). Both corrected it to *zei*, which is a correct alteration, but due to changes made in the annotation guidelines, we kept the original archaic verb conjugation if they were spelled correctly. Other differences between the two systems are the words: *haar* (her) and *bijzonder* (special). The rule-based system lacks a rule for correcting *haren* → *haar*, but it did have a rule for cases such as *hare* → *haar*. It also contains a rule for *bizonder* → *bijzonder*, but our training data for ByT5 only had one instance in it, which proved too little.

Lastly, the novel *Titaantjes* had the most significant difference, but this is due to the first case, the correction *-ie* → *hij* is a correct alteration, but again, we kept instances such as these since it is still valid for modern Dutch. Leaving this case out of the total mistakes, our best model still has seven mistakes less (70 vs 63). Other interesting cases were *metdertijd* (over time) and *koeienogen* (cow eyes). The rule-based system had a rule for correcting *metdertijd*, but again, the training-set lacked such instances. The word *koeienogen* is not corrected by the rule-based system, but since our model has seen cases for the word *ogen* (eyes), it was able to make the correct prediction.

5.4 Extended analysis

In Figure 6, predictions of sentences from the three novels can be found; here, the output is shown for the rule-based system and our best FlanT5 and ByT5 models. Since our models used silver data originating from the rule-based system, the correct spelling of the word *tooneelmenschen* is not in our model's training corpus; neither are all words marked in red for the rule-based (RB) system. However, our models are capable of correcting certain cases that the rule-based system does not. This can be because in our model's training data, words where a similar correction applies, such as *-sch* → *-s* or removal of double vowels *oo* → *o*, the system can generalize it to different cases. While FlanT5 and ByT5 are both capable of making corrections not found in the training data, we can still see that our character-based model performs better in certain situations. In sentences 3, 5, 8, 9, and 10, ByT5 corrects words FlanT5 does not, indicating that character-based models are better at making such generalizations. Still, there is one instance where FlanT5 performed better. The word *mijnheer* (sir) is normalized correctly by both the rule-based system and FlanT5, while ByT5 did not correct it.

5.5 Rule-based + ByT5

Our ByT5 model performed the best on all three novels based. However, there were cases were the rule-based system corrected words ByT5 could not fix. Combining the output of ByT5 with the rule-based system could result in even better performance. For this, we iterated over each word in each novel's prediction from the rule-based and ByT5 model. If there was a disagreement, we favored the ByT5 prediction over the rule-based one. Since ByT5 can correct more unique words, favoring ByT5 decisions could lead to better performance. In the end, this combined system did not outperform our best model, with a lower ERR score for all three novels. This was mainly due to the systems having different unique words that they predicted correctly or incorrectly. And since we only could decide on favoring one model over the other for cases of disagreement, we introduce both correct and incorrect cases. For example: if there was a disagreement about the word *bijzonder* (exceptional), and we favored the ByT5 (*bizonder*) over the rule-based (*bijzonder*) system, we would

Orig	1.	- Zelfs als die tooneelmenschen armoede willen voorstellen, is hun voorstelling altijd leugenachtig.
RB	1.	- Zelfs als die tooneelmenschen armoede willen voorstellen, is hun voorstelling altijd leugenachtig.
FlanT5	1.	Zelfs als die toneelmensen armoede willen voorstellen, is hun voorstelling altijd leugenachtig.
ByT5	1.	- Zelfs als die toneelmensen armoede willen voorstellen, is hun voorstelling altijd leugenachtig.
Orig	2.	- Het was me bijzonder aangenaam u weer te zien , m'nheer ...
RB	2.	- Het was me bijzonder aangenaam u weer te zien , mijnheer ...
FlanT5	2.	- Het was me bijzonder aangenaam u weer te zien , mijnheer ...
ByT5	2.	- Het was me bijzonder aangenaam u weer te zien , m'nheer ...
Orig	3.	Toen ik 't laatst op gindschen grond
RB	3.	Toen ik 't laatst op gindschen grond
FlanT5	3.	Toen ik 't laatst op gindschen grond
ByT5	3.	Toen ik 't laatst op gindse grond
Orig	4.	Zoo'n meisjen is natuurlijk de heldin .
RB	4.	Zo'n meisje is natuurlijk de heldin .
FlanT5	4.	Zo'n meisjen is natuurlijk de heldin .
ByT5	4.	Zo'n meisjen is natuurlijk de heldin .
Orig	5.	Toch kwamen wij , jongens van quarta , altijd 's avends op de Westermarkt om dat meisje te zien .
RB	5.	Toch kwamen wij , jongens van quarta , altijd 's avends op de Westermarkt om dat meisje te zien .
FlanT5	5.	Toch kwamen wij , jongens van quarta , altijd 's avends op de Westermarkt om dat meisje te zien .
ByT5	5.	Toch kwamen wij , jongens van quarta , altijd 's avonds op de Westermarkt om dat meisje te zien .
Orig	6.	't Eerste licht in moederoogen ?
RB	6.	't Eerste licht in moederoogen ?
FlanT5	6.	't Eerste licht in moederogen ?
ByT5	6.	't Eerste licht in moederogen ?
Orig	7.	Over het PERPETUUM MOBILE , de cirkelkwadratuur en den wortel van wortellooze getallen .
RB	7.	Over het PERPETUUM MOBILE , de cirkelkwadratuur en de wortel van wortellooze getallen .
FlanT5	7.	Over het PERPETUUM MOBILE , de cirkelkwadratuur en de wortel van wortelloze getallen .
ByT5	7.	Over het PERPETUUM MOBILE , de cirkelkwadratuur en de wortel van wortelloze getallen .
Orig	8.	Ziet dezen krans met paarlen bezet , naast de kinine-flesch .
RB	8.	Ziet dezen krans met paarlen bezet , naast de kinine-flesch .
FlanT5	8.	Ziet dezen krans met paarlen bezet , naast de kinine-flesch .
ByT5	8.	Ziet dezen krans met paarlen bezet , naast de kinine-fles .
Orig	9.	De Beij van Tunis kreeg een kolijk als hij het wapperen hoorde van de nederlandsche vlag .
RB	9.	De Beij van Tunis kreeg een kolijk als hij het wapperen hoorde van de nederlandsche vlag .
FlanT5	9.	De Beij van Tunis kreeg een kolijk als hij het wapperen hoorde van de nederlandsche vlag .
ByT5	9.	De Beij van Tunis kreeg een kolijk als hij het wapperen hoorde van de nederlandse vlag .
Orig	10.	Over het SANSKRIT , als moeder van de germaansche taaltakken .
RB	10.	Over het SANSKRIT , als moeder van de germaansche taaltakken .
FlanT5	10.	Over het SANSKRIT , als moeder van de germaansche taaltakken .
ByT5	10.	Over het SANSKRIT , als moeder van de germaanse taaltakken .

Figure 6: 10 example sentences from the test sets for the Rule-based (RB) system and our best word-based and character-based model (FlanT5 finetuned with 10k (FlanT5), pretrained ByT5 finetuned with 10k (ByT5)). Underlined words in the original (Orig) are the old spelled words, words marked red are incorrect, and words marked green are correct.

select an incorrect instance. But for the word *hardstenen* (bluestones), we favored ByT5 (*hardstenen*) over the rule-based (*hardsteenen*), we would select the correct version.

5.6 Evaluation on downstream tasks

Similar to van Cranenburgh and van Noord (2022), we report the effect of spelling normalization on two downstream tasks: parsing and coreference resolution. For parsing, we report the labeled dependency F1 score. We only report parsing scores for *Titaantjes*, since this is the only text for which we have manually corrected parse trees available. For coreference, we report mention identification performance with the span F1 score and coreference performance with the CoNLL score (Pradhan et al., 2012). We consider four versions of each text:

1. the original text
2. a normalized version produced by the rule-based system
3. a normalized version produced by the best neural system (pretrain ByT5 Books 10k)

Text	Spelling	Dep F1	Mention F1	CoNLL
Titaantjes	original	86.44	86.40	66.39
	rule-based	89.73	88.42	67.39
	neural	89.80	88.43	67.98
	manual	89.97	88.61	68.17
Max Havelaar	original		84.47	64.89
	rule-based		85.98	66.02
	neural		85.89	65.98
	manual		86.33	66.55
De Agra Schat	original		84.24	57.32
	rule-based		86.44	59.44
	neural		86.58	59.51
	manual		87.51	60.12

Table 8: The effect of spelling normalization on downstream tasks.

4. a manually normalized version

Each version of this text is parsed by Alpino, and coreference is resolved using the dutchcoref system (van Cranenburgh, 2019). Before the texts can be parsed by Alpino, some massaging of the data is required. In our setup, Alpino expects the original text with any normalizations indicated using meta annotations; see (1) for an example. Getting the data in this format requires alignment of the original and the normalized sentence, and adding the right meta annotations.[3] We perform these postprocessing steps using several ad hoc scripts; for details, see the Github repository.[4]

The results are in Table 8. The scores mostly follow the expected pattern; i.e., manual spelling normalization gives best results, followed by the neural system, the rule-based system, and finally the original non-normalized text. However, the differences between the normalization methods are quite small.

6. Conclusion

In this section, we will first answer our research questions posed in the introduction, second, we discuss the limitations of our work, and lastly, discuss possible future research.

Our first research question was: *Which type of language model yields the best performance for the task of historical Dutch spelling normalization?* We trained four different models: FlanT5, a word-based model, and three versions of the token-free model ByT5, where two were pretrained first. Each model was finetuned with two sizes of the same dataset, 5k and 10k sentences, respectively. Our best models were the pretrained ByT5 models. The model pretrained with books achieved the best results on the novels *Max Havelaar* and *Sherlock Holmes*, whereas the pretrained version with the SoNaR corpus achieved the best results on the novel *Titaantjes*. Therefore, we can conclude that for historical Dutch spelling normalization, a pretrained ByT5-small model, which has been finetuned afterward, is the best approach. However, if time and computational resources are limited, the original ByT5, pretrained on the whole mC4 corpus, still achieves good results—far better than the subword model FlanT5.

3. In future work, it might be better to train the neural system to directly produce the normalized sentence in the format which Alpino expects, but this would still require postprocessing, to ensure the output contains syntactically correct meta annotations.
4. https://github.com/andreasvc/neuralspellnorm

Our second research question was: *To what extent do the pretrained language models outperform the rule-based system in terms of spelling normalization performance?* This question related to whether language models trained on data from a rule-based system are able to beat that same rule-based system. Here we make the distinction between our word-based and character-based models. For the word-based model (FlanT5), training on the silver data did not lead to better results than the rule-based system. For the character-based models (ByT5), only the model pretrained with the BERTje books corpus and finetuned with our largest dataset could outperform the rule-based system on all three novels if we consider all four metrics. The original ByT5 version could only outperform it for the novels: *Sherlock Holmes* and *Titaantjes*, which were from the later periods (1899 & 1915). Therefore, we can conclude that a pretrained monolingual Dutch ByT5 model finetuned for spelling correction can outperform the rule-based system for 19th-century spelling normalization. However, considering the hallucinations these models have to deal with, something the rule-based system doesn't affect, the risk of having situations where the output is drastically different is something to keep in mind.

The third research question was: *Is a pretrained language model trained with silver data derived from a rule-based system able to make generalizations not found in the training data?* This question related to the capability of our trained models to correct spelling mistakes that were not present in their training corpora. Both FlanT5 and ByT5 are capable of making corrections not found in the training data. Between the word-based and character-based models, our character-based model (ByT5) is, to this end, better at correcting mistakes not present in the training data. This means that all of our trained models can make generalizations not found in the train set, where the token-free models have the upper hand.

Lastly, our final research question was: *What is the effect on downstream tasks when normalizing spelling with pretrained language models, compared to other spelling normalization methods?* We find that in most cases, normalizing spelling with pretrained language models improves performance in coreference resolution and parsing. However, the differences are small.

6.1 Limitations

Our initial research plan was only to finetune two T5 versions for 19th-century Dutch spelling normalization, but due to lack in performance, we opted to also pretrain our token-free model. But since we started pretraining later in our research, we had to limit our pretraining corpus to 2 million sentences, whereas most pretrained models use larger corpora (20+ million sentences), which can result in better performance overall. Another limitation was our computational power for pretraining. We had to rely on the Hábrók GPU cluster, but there was a time limit of 72 hours, and training with only 2 million sentences was already pushing the system with the smallest version of ByT5 available. Therefore, having access to more compute power could enable pretraining a larger version of ByT5 with more data. We also used two dataset sizes for finetuning datasets (5k and 10k), but since our 10k corpus performed better on almost all models, more training data could have given even better performance. We also saw in the extended analysis that our models lacked performance due to insufficient instances in the training data; increasing the data and ensuring the model learns more cases can benefit its generalization capability. Lastly, taking into consideration the hallucinations and the amount of data and computational costs of our models, one can argue that extending the rule-based system may be more effective. The rule-based system, as discussed in Section 2, has both manual and automatic rules; extending the latter would make the model even better without many extra resources required.

6.2 Future research

Due to the restrictions just described, future research possibilities could focus on pretraining ByT5 for the Dutch language with more data, with a corpus well above 20+ million sentences. This could

benefit not only performance for spelling normalization but also other tasks in NLP. Tasks such as entity tagging, or label classification, which for now are mainly done with encoder models (for Dutch, BERTje or RobBERT, Delobelle et al. 2020). A high-quality Dutch monolingual seq2seq model could open the doors for many more NLP tasks, such as machine translation. Other work could focus entirely on the token-free model and pretraining or finetuning a larger variant of ByT5; since it comes in different sizes (small, base, large, XL), we only focused on the smallest version due to resource limitations. Lastly, an extension to this study could be to focus exclusively on gold data rather than silver data, which could show the difference between training with automated data and the best possible data available. Or, instead of focusing on an ML approach, extend the existing rule-based system.

Acknowledgments

We are grateful to Lukas Edman for advice on pretraining ByT5, to Gertjan van Noord for assistance with Alpino evaluation, and to three anonymous reviewers for feedback on the paper.

References

Guillaume Chiron, Antoine Doucet, Mickaël Coustaty, and Jean-Philippe Moreux. 2017. ICDAR2017 competition on post-OCR text correction. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 1423–1428.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. BERTje: A Dutch BERT Model. arXiv:1912.09582.

Pieter Delobelle, Thomas Winters, and Bettina Berendt. 2020. Robbert: a dutch roberta-based language model.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Allen Kim, Charuta Pethe, Naoya Inoue, and Steve Skiena. 2021. Cleaning dirty books: Post-OCR processing for previously scanned texts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4217–4226.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

Anneke M. Nunn. 2006. *Dutch orthography: A systematic investigation of the spelling of Dutch words*. Ph.D. thesis, Katholieke Universiteit Nijmegen.

Fabian Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395.

Maja Popović. 2017. chrF++: words helping character n-grams. In *Proceedings of the Second Conference on Machine Translation*, pages 612–618.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40.

Martin Reynaert. 2008. All, and only, the errors: more complete and consistent spelling and OCR-error correction evaluation. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*.

Ineke Schuurman, Véronique Hoste, and Paola Monachesi. 2010. Interacting semantic layers of annotation in SoNaR, a reference corpus of contemporary written Dutch. In *Proceedings of LREC*, pages 2471–2477.

Noam Shazeer. 2020. GLU variants improve Transformer. arXiv:2002.05202.

Elizabeth Soper, Stanley Fujimoto, and Yen-Yun Yu. 2021. BART for post-correction of OCR newspaper text. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 284–290.

Andreas van Cranenburgh. 2019. A Dutch coreference resolution system with an evaluation on literary fiction. *Computational Linguistics in the Netherlands Journal*, 9:27–54.

Andreas van Cranenburgh and Gertjan van Noord. 2022. OpenBoek: A corpus of literary coreference and entities with an exploration of historical spelling normalization. *Computational Linguistics in the Netherlands Journal*, 12:235–251.

Andreas van Cranenburgh, Sara Veldhoen, and Michel de Gruijter. 2022. Textual features and metadata for DBNL novels 1800-2000. Zenodo data set.

Rob van der Goot, Alan Ramponi, Arkaitz Zubiaga, Barbara Plank, Benjamin Muller, Iñaki San Vicente Roncal, Nikola Ljubešić, Özlem Çetinoğlu, Rahmad Mahendra, Talha Çolakoğlu, Timothy Baldwin, Tommaso Caselli, and Wladimir Sidorenko. 2021. MultiLexNorm: A shared task on multilingual lexical normalization. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 493–509.

Gertjan van Noord. 2006. At last parsing is now operational. In *TALN06. Verbum Ex Machina. Actes de la 13e conference sur le traitement automatique des langues naturelles*, pages 20–42.

Nynke van 't Hof, Vera Provatorova, Mirjam Cuper, and Evangelos Kanoulas. 2022. OCR error detection and post-correction with Word2vec and BERTje on Dutch historical data. Abstract presented at DHBenelux 2022.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498.

Appendix A. Historical Dutch Spelling Normalization Annotation Guidelines

This annotation guideline describes the rules for correcting silver data or annotating historical Dutch 19th-century texts. These rules only apply to changing the spelling of the text, not rephrasing the sentence, changing the grammar, etc. The spelling corrections take the form of instructions to the Alpino parser. There are five instructions relevant for spelling normalization: `@alt, @alt_mwu, @mwu, @phantom`, and `@postag`. For the full documentation, see the section on bracketed input in the Alpino User Guide[5]

1. When two words are merged into one, combine the words and place them after the `@alt_mwu` placeholder (right side); the old spelled words are added after (left side): [@alt daarom daar om]
2. When one word should be split into multiple words, apply `@alt` to the most important word, and insert the other words with `@phantom`:
   `[ @phantom aan ] [ @phantom te ] [ @alt spreken aantespreken ]`
3. When the silver spelling correction offers multiple possibilities, pick the correct one:
   `de aralia's en palmen van [ @alt ~de~den den ] corridor`
   `de aralia's en palmen van [ @alt de den ] corridor`
4. In case of doubt about any word or phrase, consult the Historical Dutch dictionary[6] to check for the correct instance.
5. Leave the hyphenation in hyphenated words unchanged, (e.g. twee-en-negentig, op-zichzelf). But do correct single words which are written in old Dutch, (e.g., Noord- Duitschland → Noord-Duitsland, kampioen-athleet → kampioen-atleet)
6. Normalize adjectives ending in -en, for example: gespierden → gespierde
7. Normalize grammatical cases, for example: uwen, uwer, uwe → uw; onzen → onze
8. Keep archaic verb conjugation of words if the spelling is correct, for example: gij kwaamt, gij zijt, hij zeide
9. Remove circumflex accents; for example: broêr → broer; weêr → weer; elkaâr → elkaar
10. Correct words in French spelling to Dutch alternatives, if possible. For example: billardballen → biljartballen; million → miljoen

5. https://www.let.rug.nl/vannoord/alp/Alpino/AlpinoUserGuide.html
6. Woordenboek der Nederlandsche Taal (WNT): https://ivdnt.org/woordenboeken/historische-woordenboeken/