LLMs for OCR Post-Correction

Martijn Veninga
s1803727
University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
July, 2024

Abstract

In this thesis, I examine the use of Large Language Models (LLMs) on the task of Optical Character Recognition (OCR) post-correction. OCR post-correction is the task of correcting mistakes in optically identified texts from images or documents. Publicly available OCR engines are limited in accuracy and correcting their output can lead to better performance of downstream tasks.

Pretrained LLMs exhibit an understanding of language which can be exploited to correct mistakes in OCR output, but for good performance fine-tuning of the models is needed. In this thesis I show that fine-tuned versions of the ByT5 LLM, a publicly available character-level LLM, are able to correct mistakes in OCR text in a language it was pretrained on better than a state-of-the-art method can. Preprocessing techniques are shown to impact the capability of the language models to correct OCR errors. ByT5 models achieve the highest Character Error Rate (CER) reduction rates when using lowercasing as well as removing strange characters. Context length is also shown to have a strong impact on the effectiveness of the models. The best context length was found to be 50 characters, with longer and shorter context lengths having worse CER reduction and worse F1 scores. I also show that few-shot learning is not able to teach a generative LLM to correct OCR text without fine-tuning the model. Future research can investigate the potential increase in effectiveness of fine-tuning larger language models on the post-OCR error correcting task.

1 Introduction

1.1 What is OCR

Optical Character Recognition (OCR) is a method of extracting the text data from images of documents. The process is often required in settings where a digital version of the text is not available. This occurs when documents are scanned from physical copies, or when digital documents containing text do not have a computer-readable textlayer in them. OCR techniques have been used for decades now to process large amounts of scanned documents [9, 20]. State of the art models based on deep learning methods are efficient and achieve high accuracies in the OCR task [12]. Still, many errors remain in the output of OCR techniques which lead to lower performance of downstream natural language processing tasks like text summarization, part-of-speech tagging and named entity recognition [12, 25, 28].

1.2 Challenges in Modern Documents

Documents originating from computers normally contain an image layer and a text layer. The image layer is the physical appearance of the document. The text layer is a hidden information layer included in documents which separately from the image shown on screen encodes information on the positioning of text on the page. This information is used by computers to read the contents of the files. When this textlayer has been lost, optical techniques are required to get the text from the image layer of the documents. The problem we see in industry is that some documents which originated from computers have lost their text layer, and got degraded by noisy copying and compression techniques. Some examples of lossy compression techniques are Chroma Subsampling, where the color component of images is sampled at a lower rate than the brightness component to reduce storage need while achieving similar image quality to the human eye, and the K-Means algorithm, where colors are grouped into K clusters reducing the colors in an imperceptible way [14]. These techniques, coupled with the loss of the original textlayer, result in documents that are difficult to OCR perfectly, leading to character errors.

1.3 Post-correcting

As the output of OCR systems is not perfect, improvements can still be made. One method of doing so is to improve the primary OCR technique. Another path to improvement is by using post processing. Historically, post processing of the OCR output has been an effective technique for improving the initial results obtained from the OCR system [15]. Commonly used automatic techniques include merging the outputs of several OCR engines together based on a voting scheme [22], or using n-gram language models to detect errors on character level to then suggest candidate corrections on word level [8]. Google have used a form of crowd-sourced post-OCR correction through their ReCAPTCHA service [45]. ReCAPTCHA is used by websites to protect against bots. This is done by requiring users to transcribe two words shown in images. One of them has a known text and is used for determining if the user is human, the other word does not have a known text and will at a later stage be determined by majority vote. Users are not aware of the fact that one of the words is unknown, so they fill in both words correctly based on their own best effort.

The effectiveness of these techniques has been promising, but improvements can still be made. The final accuracy of these methods is not 100%. The popular Tesseract OCR engine was shown to have a Character Error Rate (CER) of 0.4% depending on the quality of the document that is processed. Still, reducing the number of character errors can yield better output in downstream tasks such as accessibility readers and language processing tasks [12, 25, 28]. Schaefer et al. show that their two-step error correcting method can reduce OCR errors by 20% [35]. Still, humans reading through the output of these OCR techniques will still be able to find misplaced or missing characters, and with their understanding of language are able to improve the OCR accuracy. Manually correcting all documents is expensive and slow and thus not a viable solution for real-time low-cost applications. By automating the task, notable improvements to the performance of down-stream tasks like part-of-speech tagging and named entity recognition can be achieved compared to using the uncorrected OCRed texts, while keeping costs low and speed high [28]. Hegghammer et al. show that named entity recognition quality is impacted by the quality of OCRed text [12]. Huynh et al. similarly show that OCR post-correction is able to improve the quality of named entity recognition when applied to already reasonably accurate OCR output [15].

1.4 Large Language Models

In recent years a new automated learning-based technique called Large Language Models (LLMs) has emerged boasting a strong understanding of language, opening the door to automated human-like understanding of texts for post-OCR error correction.

LLMs first gained considerable attention through the seminal paper "Attention is all you need" [43]. The paper proposed an architecture of language models which does away with recurrence as used in LSTM and RNN models, instead using only the attention mechanism to draw global dependencies. The attention mechanism consists of a mapping of a query, and key-value pairs to an output. The attention mechanism allows transformers to take into account and weigh the relevance of each part of the complete context for any section of the prompt. It also allows for greater model sizes, as it solves the parallelization issue LSTMs and RNNs had [43]. With increased size, the performance of these models keeps increasing, allowing for super large models to become especially good at understanding language [4, 16]. More recently, the online available ChatGPT became popular because of availability and its ability to perform a plethora of language-based tasks. The model shows an understanding of language which generalizes well to tasks like coding, sarcasm detection, emotion detection and question & answering among others [17]. Publicly available models like Llama show similar language understanding [49]. This understanding of language might be useful in improving OCRed text. Rigaud et al. showed that LLMs can leverage their basic understanding of language in order to improve the OCRed text [32].

ChatGPT and Llama models are of very large size. The GPT3 model of OpenAI has 175 billion parameters, the smallest Llama model has 7 billion parameters [41]. The size not only leads to large inference times, but also makes fine-tuning very challenging. Fine-tuning these models needs to be done on servers with depending on the model 100s of gigabytes of VRAM, which is costly and not readily available to students. Still, these models can be tested on new tasks by using few-shot learning.

Using smaller LLMs than Llama and ChatGPT results in faster inference, and through fine-

tuning of such a model, previous researchers have achieved state-of-the-art performance in the ICDAR 2019 competition for post OCR text correction [33]. In the 2019 competition, the first publicly available BERT model was used. Since then, newer architectures have come out that may lead to better performance than the model used in the 2019 competition.

For the input, the text is split up into tokens. Most models use sub-word tokens, meaning that the input is split into blocks of several characters which are given a unique integer representation for the model. Some models use character-level tokenization, meaning that each individual character has an integer representation. Sub-word tokenization works well and is used in many of the highest performing language models available. Mielke et al. note that tokenization of larger sections of text leads to faster inference [24]. Using character-level representations however allows for the model to take note of specific one-character modifications between strings. This can be especially useful in the context of OCR post-correction, in cases where single characters are misidentified.

1.5 Proposed Solution

I propose a post-OCR text correction technique that leverages the language understanding of modern LLMs to detect mistakes and suggest corrections in modern documents processed with the Tesseract OCR engine [40]. For comparison to other methodologies, the same technique is tested on old documents from the ICDAR dataset. OCRing large amounts of modern customer documents is done using open-source OCR engines like Tesseract, whose results might be improved by the solution provided here. The LLMs tested in this research include fine-tuned versions of the character-level LLM Byt5 trained on OCRed and ground-truth text, as well as the general generative LLM Llama 7B [41, 48]. For ByT5, the models ByT5 small and ByT5 base are used which consist of 300 million and 580 million parameters respectively. The Llama 7B model consists of 7 billion parameters. Llama 7B was selected because it is one of the best performing publicly available generative large language models that is small enough to run locally [41]. ByT5 is selected because of its character-level architecture which can be uniquely beneficial in spotting the character-level mistakes in OCR output [48].

1.6 Research Questions

In creating this LLM based post-OCR text correction technique, several pre-processing methods are employed in experiments to test if their application aids or hinders the ability of the LLM to correct the text. Lowercasing of the input text and removal of special characters are examined.

Researching the capability of post-OCR error correction of LLM models is done in this thesis to answer the following research questions:

- RQ1. Can character-level LLMs perform better in correcting OCR output than state-of-the-art non-pretrained language model based methods?
- RQ2. Do finetuned character-level LLMs outperform the generative LLM Llama in post-correcting OCRed text?
- RQ3. How do preprocessing techniques impact the effectiveness of the LLM post-OCR correcting techniques deployed?

2 Related Work

In their 2011 summary, Volk et al. note three methodologies for OCR text improvement: improving the input images, improving the OCR system or OCR post-processing [44]. In this thesis I focus on an automated OCR post-processing approach.

The topic of post-OCR error correction has been the subject of various competitions and a large amount of research. Non-transformer based methods have been shown capable of improving OCRed text [23], but in recent years transformer based methods have achieved slightly better results [33]. Although the two approaches reach similar performance, the potential for further improvement is still large in transformer based methods as more sophisticated models become available to researchers.

In this section, the history of OCR post-correction is described. The first subsection investigates the methodologies used before transformer models were invented. Next, the ICDAR competitions on OCR post-corrections are discussed [5, 33]. Next the ensemble of sequence-to-sequence model approach by Ramirez et al. is described [30]. Lastly, a history of large language based text correction methods is given.

2.1 Pre-transformer Post-OCR Correction Methods

Before the advent of transformers and large language models, post-OCR text correction was done using a variety of techniques. Automatic methods include those that only take into account the specific word being corrected (merging of OCR outputs, lexical approaches), but also context-dependent techniques involving statistical or neural-network based language models [28].

Dictionary based approaches involve building a dictionary of n-grams that are found in known texts. The dictionary is used to look up the text resulting from noisy inputs like OCR, to find out if the text is known and likely to appear. In the case that it is not, the program detects it as a potential error, but the correcting task is left to the user [18].

An unsupervised corpus-based method that uses high quality lexicons to automatically improve OCR errors was developed by Reynaert et al. [31]. The method calculates Levenstein distances between known words and the words found in the input text to correct the OCR text.

Schaefer et al. [35] proposed a two-step error detection and error correction methodology based on bi-directional LSTM models. The first step, detector, is an error detection mechanism which identifies potential OCR errors and forwards them to the second step of the process. This second step is the error correcting mechanism called the translator which predicts the correct text given its input. By using the two-step approach, non-erroneous text is filtered out and skipped by the correcting model, which reduces the number of false positives. Both the detector and the translator are bi-directional LSTM models trained on their dataset. They show that their methodology can reduce character error rate by up to 20% in text coming from Tesseract OCR engine. Their method shows good performance of improving OCRed text, without using large language models.

2.2 ICDAR Competitions

In 2017 and 2019, competitions on post OCR text correction were held by ICDAR [5, 33]. In the 2017 competition, the best solution used a character level machine translation based on the 4 character context surrounding each character [27]. Most recently, in the 2019 competition, the best solution was a multilingual fine-tuned BERT [6] as a character sequence machine translation model.

Since 2019, extensive research has been done on the capabilities of transformer models, leading to advances in model language understanding. New language models have since been proposed that improve upon BERT in several ways, such as roberta, deberta and debertav3 [10, 11, 21], but generative AI models have also been improving. Meta released their Llama generative AI model [41] as open source. These new models show improved performance on language processing tasks, indicating that their performance on the OCR post-correction task may well also be better than the older models.

2.3 Ensemble Sequence-to-Sequence

Ramirez et al. [30] proposed an ensemble of trained sequence-to-sequence models for OCR post-correction in 2022. In their paper they show that combining the results of a large set of sequence-to-sequence models trained on the target task is able to reduce character error rates on the ICDAR data that matches the state-of-the-art reported during the original competition in 2019. These models are not pre-trained on large sets of text data, but instead are trained directly on the target task.

In their method, Ramirez et al. split up the input text into shorter windows to separately correct. The output from the model on these windows is then separately combined using a beam search algorithm to find the most likely intended output from the model. Using this splitting and then reassembling approach, large documents can be processed without the need for training sets with many documents, or very high memory compute units. Figure 1 illustrates their process.

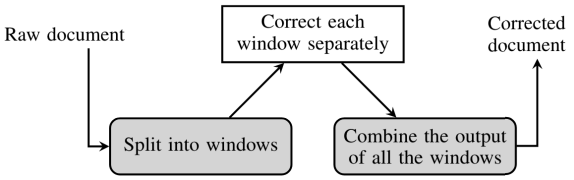


Figure 1: Description of Steps in the Methodology Proposed in Ramirez et al. (My Baseline)

2.4 Large Language Models for Text Correction

Alikaniotis et al. [2] showed in their paper that large language models are able to correct grammar mistakes. They used BERT, GPT-1 and GPT-2 along with different preprocessing techniques to correct source text containing grammar mistakes.

They show that the GPT models significantly outperform BERT on the grammar correction task.

Kuznetsov and Urdiales [19] use a transformer architecture for spelling-correction. The transformer architecture is the basis of the LLMs discussed earlier. They show that transformers are able to learn from a dataset of spelling mistakes, to be able to correct spelling mistakes in real-world settings. Just like spelling errors, OCR errors often involve single-character replacements. Therefore, techniques used for correcting spelling errors may also be useful for fixing OCR errors.

The transformer architectures used by Kuznetsov and Urdiales are not trained to fully understand language, but solely leverage the attention-mechanism for better context understanding. Pre-training transformer models gives them knowledge of languages and biases them towards proper language generation [6]. Thus using a pre-trained transformer model could be effective in correcting text inputs. Stankevivcius et al. [37] took this approach in their paper, fine-tuning a pre-trained ByT5 character-level transformer model to correct words with missing diacritics. They show that their methodology was able to detect missing diacritics in over 98% of cases, vastly outperforming dictionary based-techniques and rivalling state-of-the-art in the task. This was achieved whilst being trained on much less data than the state-of-the-art methodologies. They also showed that their model was able to detect missing diacritics in words not seen in the training dataset with 76% accuracy.

2.5 Generative Models for Text Correction

Sutter et al. [39] showed that using a pipeline of detecting mistakes in text, combined with prompt-engineering on Llama2, they were able to correct grammatical errors. Their prompt uses very particular instructions to make the model output only the answer to correct the input, without extra explanations or contexts. The prompt they used uses HTML-style start and end tags to enclose the expected answer.

This is an example of their prompt:

> Reply with a corrected version of the input sentence delimited by <input> </input> with all grammatical and spelling errors fixed. If there are no errors, reply with a copy of the original sentence. Output the corrected version of the sentence delimited by <output> </output> tags directly without any explanations. Please start: <input>She like to walks her dog</input> <output>

Using this method, they showed that the model was able to correctly identify and correct grammar mistakes more effectively than GPT3.5, but still worse than state-of-the art in grammatical error correction solution GECToR [29, 39].

3 Datasets

3.1 ICDAR dataset

This thesis makes use of two datasets. The first one is the 2019 ICDAR competition on post-OCR text correction dataset, which contains aligned OCR and ground truth data for multiple languages including English, Dutch and German [33]. This dataset is freely available for non-commercial use. The dataset does not come with the original documents which were OCRed, but for the purpose of investigating the capability of LLMs to correct OCRed text the original OCRed text and a gold standard is sufficient. For this thesis, the English portion of this dataset is used, in order to compare the results with those obtained in literature.

The ICDAR dataset is created from old documents, including 18th and 19th century documents. These documents contain language that is not representative of the language found in current age documents. Some examples of the text found in this dataset is given in Figure 2. The dataset is useful for evaluating the performance of this method against a baseline method in the literature, but as the trained model will tend to make the text similar to that found in the training set i.e. 18th and 19th century language, the final model would not be very useful for usage on modern documents. The ICDAR dataset is thus used for comparison against the baseline.

Because the character error rate of some of the documents in the ICDAR dataset was very high, above 50%, some of the entries were filtered out for more realistic results. The final dataset size in words and characters as well as the character error rate before correction for the train, evaluation and test splits is given in Table 1. The base character error rate for the different splits of the ICDAR dataset is around 5.1%.

3.2 Custom Modern Documents Dataset

In order to get access to more relevant current-age documents, a collaboration with a local tech

The beste knyght.and the @moost manly man As myne
wait your coming at the Greyhound Tavern in Palace-Yard
VIRGINIA WATERS. The Royal Pavilion THE UPPER LAKE, With the Frigate
Have found the fame your shores refuse; Their place of birth alone is

Figure 2: ICDAR: Examples of Ground Truth Text

Dataset	Characters	Words	CER (%)
ICDAR - train	109002	19561	5.13
ICDAR - eval	33421	5950	5.10
ICDAR - test	21209	3715	5.15
Custom - train	704449	99970	5.44
Custom - eval	175796	25122	5.55
Custom - test	98264	14304	5.42

Table 1: Dataset Overview

company was started. This company specializes in providing document anonimization services using LLM anonimization suggestions. For training this LLM and testing their product, they have acquired a set of 5000 PDF documents. They have been scraped from a Dutch website posting spatial planning of Dutch municipalities [26]. The same type of documents are often seen needing anonimization in practice. These documents are scanned by the free to use Tesseract OCR [40] to generate the OCRed text which contains character errors due to the imperfect nature of the OCR method. The documents also include a ground-truth textlayer.

Originally, the idea was to match up this ground-truth textlayer embedded in the PDF documents with text obtained from OCRing those same documents, but whilst analyzing this data, it was found out that the order of the text obtained from the embedded text layer and the text in the OCR layer was not the same. The reason for this is that the embedded text layer contains the text in the proper document order, based on the tags of the document, whereas the OCRed textlayer only takes into account the spacing of text on the page, not the tags given to the text. This problem with text order is an issue for training the correction models, as they require the text to be aligned properly. A new methodology was thus used to create this dataset.

To get the custom dataset of recent documents, the original text was scraped from the ruimtelijkeplannen documents. This text represents the ground truth data. After scraping this data, a new PDF document was made which only contains this ground truth data in order on the pages. Using a similar approach as Ding et al. [7] to reduce the quality of the PDF, this new PDF document was artificially degraded using white noise. OCRing the PDF document yielded noisy OCR data to be used for training the models. The entire process of generating the dataset is summarized in Figure 3.

The final dataset size in words and characters as well as the character error rate before correction for the train, validation and test splits are given in Table 1. The train split consists of 72% of the dataset, the validation split consists of 18% of the dataset and the test set consists of 10% of the dataset. The base character error rate for the different splits of the custom dataset is around 5.4%.

The quality of OCRed output text depends not only on the OCR engine used, but also the quality of the input text. Depending on how clear and high-resolution the input images are, and how distorted the input is, the character error rate of the same OCR engine can be different. For the datasets used in this thesis, the initial OCR character error rate is just above 5%. That is the error rate seen in the ICDAR dataset, and the custom dataset was made to be similar so that the results can be compared.

4 Data Preparation and Cleaning

Model training was attempted with whole pages of text data, but this proved ineffective. Both the ByT5 and Llama models "go off the rails" with such large text inputs, yielding an output where the first bit of text corresponds to corrected text, but the last section becomes made up text that does not correspond to the original OCRed input.

Because of this, a dataset was required that can be split up into smaller sections of 10 to 100 characters at a time. To split the dataset like this, an alignment is needed. For the ICDAR dataset, this alignment was already given in the original dataset. For the custom dataset of modern documents, the OCR and ground-truth text is aligned so that any subsection of the two strings would correspond to the same piece of text. The alignment was performed using a dynamic programming approach

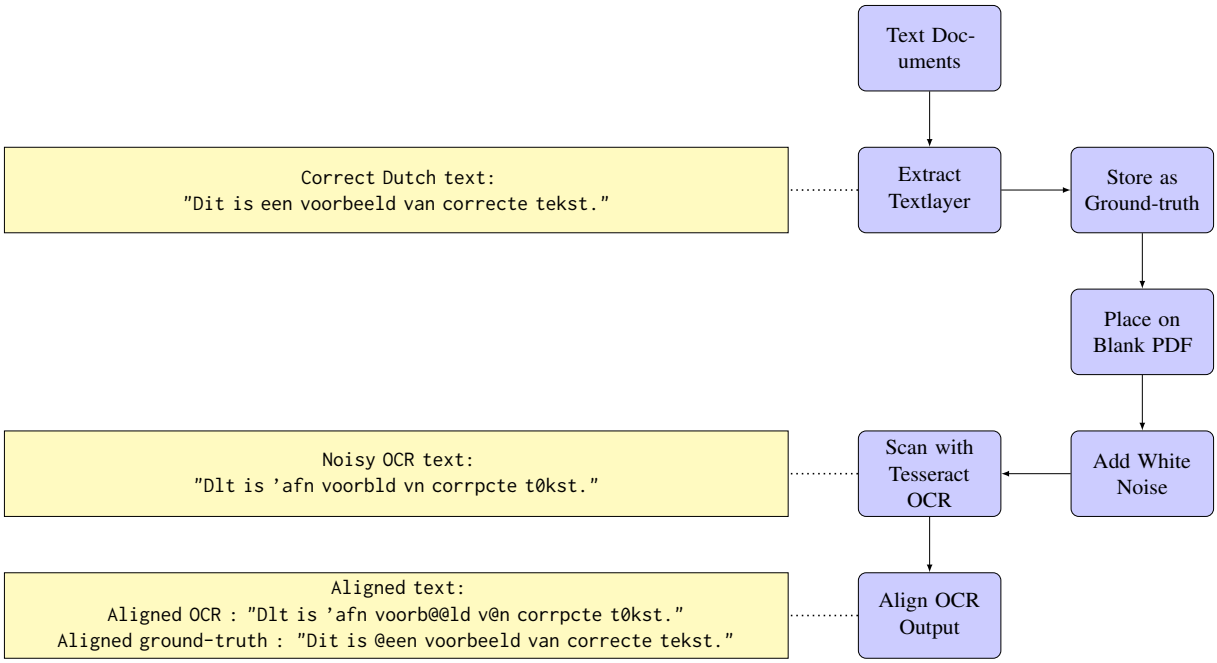


Figure 3: Custom dataset creation from original Dutch documents | Extra high CER used for illustration

which is explained in the following section.

4.1 Sequence alignment

Alignment of the strings is done by inserting gap tokens ("@") into the strings in such a way that most characters from the original strings are index-matched, while using the fewest modifications possible.

Aligning the two strings was done using a dynamic programming approach commonly used in DNA alignment [34]. Dynamic programming is useful for this approach as it is an efficient methodology for aligning the sequences. Other methodologies like an exhaustive search of all the possible ways of alignment grow exponentially with the number of characters in the strings, where this dynamic programming approach only scales with O(M*N) where M and N are the lengths of the sequences to be aligned. This makes it feasible to align large enough sequences for the task of generating the dataset.

The dynamic programming approach is done in 6 steps:

1. Creating the dynamic programming matrix. The matrix will contain the final solution and is of size M * N, where M and N are the sizes of the input strings that are to be aligned. Filling this table is where the overall time complexity of O(M*N) comes from. The table is filled with zeros, representing the cost to align sub-sequences of the original strings.

2. The dynamic programming table is filled. The table is iterated over from the top left to the bottom right. At each step, if the character is the same in both strings, the penalty value from the top left diagonal is copied in the current cell, representing no action taken. If the characters are not the same, the minimum value of the cell on the left and the cell on the right is taken along with a penalty of 1, representing the modification of the string to align the sequences needed.

3. The aligned sequences are built. In this step, the algorithm is started from the bottom right of the dynamic programming table and builds back up to the top left. At each step, if the characters are the same, they are added to the aligned sequences.

If the characters are not the same, the algorithm checks which of the cells on the left, top left and top has the lowest penalty value. The penalty values given in the dynamic programming table indicate whether the characters are to be left unaligned (in the case that the top left diagonal cell has the lowest penalty value), whether the first sequence should include a gap token (when the cell on the left has the lowest penalty value) or when the second sequence should include a gap token (when the cell on the top has the lowest penalty value).

4. When either of the sequences still has characters left but the other does not, the remainder is filled with gap tokens ("@").

5. The resulting recreated aligned strings are built up from the end of the strings first. The final results are thus reversed in order to get the strings back in their original direction. These reversed strings are the final aligned sequences.

4.2 Misaligned documents

At times, the alignment algorithm fails to create a proper alignment between the OCR and ground-truth data. These misaligned inputs need to be removed from the dataset, as they do not train the model to perform better OCR correction. Misaligned texts are identifiable by there being a much larger character error rate compared to properly aligned documents. Properly aligned texts tend to have an error rate below 15% in the custom dataset, where misaligned documents have a character error rate above 50%. The dataset was thus filtered to only contain input strings with a base character error rate below 50%.

4.3 Documents with high number of digits

The original custom dataset includes some documents that contain a high number of digits from tables and the like. These mostly random numbers do not contain much information for the model to train on as their order is more or less random. To prevent the model from fine-tuning on this random data, the documents containing more than 15% digit characters were removed from the document set. Table 1 shows the statistics for the custom dataset after this initial cleaning step was performed.

5 Dataset Preprocessing

5.1 Input Lengths

The LLM models used in this thesis have shown problems with learning from large input lengths. Requesting large inputs to be corrected leads to hallucination by the model, reducing the error correction effectiveness. Because of this, the strings in the dataset were split into smaller pieces for training. Different input lengths were tested to find out what balances the extra context achieved by a larger input length with without leading to hallucination of the models. For this, input lengths of 10, 25, 50 and 100 characters were tested. This was done by splitting the aligned OCR and GT texts into smaller chunks depending on the number of characters for the input being tested. We can be sure that the OCR and GT texts still correspond after because the texts were aligned before splitting.

5.2 Text Preprocessing

In order to reduce complexity of the machine learning task, a variety of preprocessing techniques are tested. These techniques lead to lower variability in characters and sets of characters, potentially leading to better post-OCR error correction through a decreased search space. At the same time, these preprocessing techniques lead to reduced information content of the text context. This could also have adverse effects on the capability of the models to correct the OCR errors. This is why tests are ran both with and without these preprocessing techniques, to investigate the effect of them.

5.2.1 Lowercasing

Lowercasing of OCRed text as well as the ground-truth text can be used to reduce the distinct number of characters and combinations of characters that the model has to learn and choose from. This technique thus greatly reduces the search space of the model, but it comes at the cost of losing the information of which words were capitalized, which often correspond to personal and street names, company abbreviations and the like. It may thus be more difficult to distinguish erroneous OCR output from names which look like common words, as the names would not be identifiable by their capitalized first letter anymore.

5.2.2 Strange Symbols Removal

Removing strange symbols could help the model by denoising the output of the OCR engine. Oftentimes when it is not possible for the OCR engine to detect which characters are present in the text, a series of quite unusual characters is produced. The question is whether this series of characters contains enough useful information about the true text that it is worth keeping them. Removing these characters is thus tested as method of reducing the search space while removing noise from the input of the model.

6 Methodology

6.1 Error Correction Models

6.1.1 ByT5

For the character-level LLM model, the ByT5-small and ByT5-base models are used, which have 300 million and 580 million parameters respectively. These models have previously shown capability of being fine-tuned for sequence-to-sequence translation tasks. By framing the post-OCR error

correction task as a sequence-to-sequence task, we can fine-tune ByT5 to correct OCR errors. This model is also available in larger versions of 1.2 billion, 3.7 billion and 13 billion parameters, but fine-tuning these very large versions was not feasible on the resources available for this thesis.

10% of the dataset is reserved for evaluation in the test step. For training, the rest of the dataset is split into an 80% and a 20% subsection. 80% of the data is used during the training process, whilst the remaining 20% remains unseen by the model and is used for the validation step of the training process. I opt not to use a k-fold cross validation technique, as it would require fine-tuning the ByT5 model once for every fold, which is not feasible in the time of this thesis project.

The models were fine-tuned using an early-stopping approach. In this way, the models were allowed to train for as long as the validation loss was still decreasing, ending with a model that balances input data fit and generalization to achieve the best results. This early stopping mechanism resulted in two to 5 epochs trained per model per dataset. The learning rate used was 0.0003. Input data was grouped into batches of 20, and in order to get a higher artificial batch size, gradient accumulation was used of 16 steps. The gradient accumulation adds together the loss values for the individual batches to achieve the same results as a batch of 320 would, but without requiring the same VRAM levels.

6.1.2 Llama

As alternative to the character-level ByT5 model, a larger generative AI model is also tested for performance in this task. Through zero-shot and few-shot learning, this type of model is able to perform a wide variety of language tasks. 0-shot learning involves explaining the task to the generative AI, without showing any examples of how to perform the task. In few-shot learning, some examples of the task correctly being performed are given [1]. The Llama 7B model is steered through prompt-engineering to take in a couple sentences of OCRed text, and return the expected original text in a standardized format [46]. No training data is needed for this model, as the 0-shot and few-shot learning methodologies depend only on the input data to the model to be able to perform the language task.

Testing of this model is done on the same 10% subsection of the dataset that was reserved as the test set for the ByT5 model.

In the first iteration of using Llama, the following prompt was used:

> You are a language model tasked with correcting OCR text. OCR text contains errors as the algorithm sometimes misidentifies characters, misses characters or adds characters where they were not originally. Use your understanding of language to find these errors and correct them.

This prompt leads to an unstructured output, as the model is not given a specific structure to answer in. Besides the corrected output, a lot of extra output is given explaining the corrections made and that the model understands the task. This output is not wanted and we want to filter it out. To do so, we specifically tell the model to leave out any outside explanation:

> You are a language model tasked with correcting OCR text. OCR text contains errors as the algorithm sometimes misidentifies characters, misses characters or adds characters where they were not originally. Use your understanding of language to find these errors and correct them. In your response, do not give an explanation of the task or an explanation of the output. Only give the corrected text.

Although we specifically instruct the model not to give extra explanations, the output still contains unwanted extra information about the corrections. Asking the model to give the answer contained in specific tags as explained in [46] helps, as we can later filter the answer to only be the content in-between these tags. The prompt is thus modified once more into the following:

> You are a language model tasked with correcting OCR text. OCR text contains errors as the algorithm sometimes misidentifies characters, misses characters or adds characters where they were not originally. Use your understanding of language to find these errors and correct them.
>
> In your response, do not give an explanation of the task or an explanation of the output. Only give the corrected

text. Give your answer in-between <corrected> and </corrected> tags like would be seen in HTML code.

Example 1:

input: "is er vanuit de ecologie van het opperviaktewater verbetering"

output: "<corrected>is er vanuit de ecologie van het oppervlaktewater verbetering nodig</corrected>"

Example 2:

input: "functies in het betrelfende gebied. om aan deze doelen structure! te kunnen voldoen,"

output: "<corrected>functies in het betreffende gebied. om aan deze doelen structureel te kunnen voldoen,</corrected>"

Using this prompt, the model consistently outputs the answer in-between the tags given, so the answer is taken from this through a regex search.

6.1.3 Baseline model

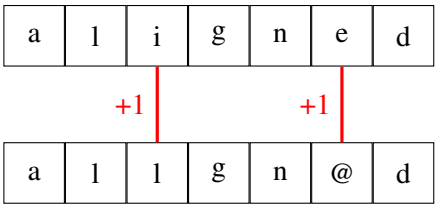
A comparison is also made with non-LLM based post-OCR error correction techniques. The methodology of [30] is implemented consisting of an ensemble of sequence-to-sequence models.

Ramirez et al. published their full working code to GitHub[1], enabling the usage of the models on the datasets created for this thesis. The models are first trained using the training dataset, which is done on each of the individual datasets tested in this thesis. The final output of the training process is tested on the 10% subsection of the dataset designated as test set. This gives the baseline performance for the LLM-based methodology to beat.

6.2 Performance measures

Performance evaluation of the proposed methods in combination with their pre-processing techniques is done using an OCR-specific metric called Character Error Rate (CER) reduction, the general precision, recall and F1 metrics. The metrics are calculated for the performance of our fine-tuned models, as well as for the baseline methodology trained on the ICDAR and custom datasets, for a full comparison of the methodologies.

CER is a metric based on the number of characters that were incorrect in the OCR output. This is calculated based on the number of characters that are different between the ground truth text and the OCR output text. By comparing the CER before the post-OCR error correction and after the post-OCR error correction, the effect of the correction method can be quantified. An example of CER calculation is shown in Figure 4.



Total Errors: 2 errors
Total CER: 2/7 = 28%

Figure 4: Character Error Rate Calculation

Another method of determining the performance of the post-OCR error correction method is through precision, recall and f1. Precision is the proportion of correctly corrected characters to the total number of characters corrected. Recall measures the proportion of correctly corrected characters as a proportion of the total number of errors present in the OCR output. The F1 score measures the harmonic mean between the precision and recall metrics, giving a balanced measure of the two. An example of the calculations are given in Figure 5

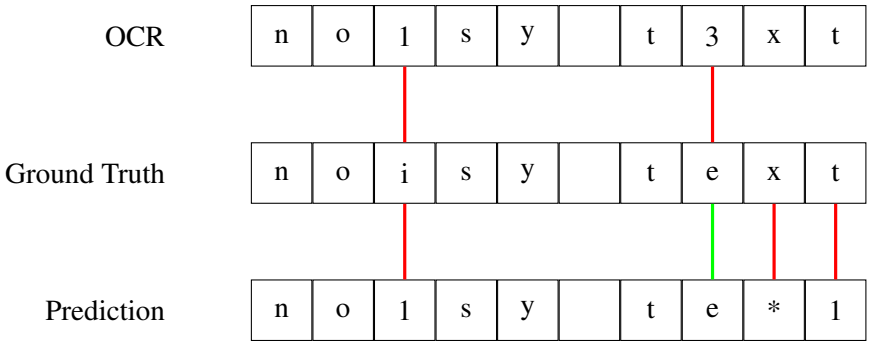
6.3 Resources

LLMs require vast resources to run. The university supplies a shared Jupyter space for students containing the NVIDIA A16 GPU with 64GB or VRAM. A private machine learning computer with a 24GB NVIDIA RTX 4090 GPU is also available through the collaboration with the local tech company. Because the RTX 4090 GPU is faster in training and inference than the university's Jupyter space, that GPU was used in the training and evaluation steps of this thesis.

7 Results

After finetuning with and without preprocessing techniques applied, the baseline and ByT5 models were tested on the test section of the custom and ICDAR datasets. The summary of these results are shown in tables. Table 2 shows the CER improvement of each configuration on each dataset. Table

[1] https://github.com/jarobyte91/post_ocr_correction



Precision (total correctly edited / total num character edited): $\frac{1}{3} = 0.33$
Recall (errors corrected / total original errors): $\frac{1}{2} = 0.5$
F1 Score: $2 \times \frac{Precision \times Recall}{Precision + Recall} = 0.4$

Figure 5: Precision, Recall, and F1 Score Calculation

3 shows the calculated F1 scores. Tables 4 and 5 show the recall and precision scores respectively.

Higher character error rate reductions are seen in the results of the fine-tuned ByT5 models for the custom dataset. For the ICDAR dataset the higher character error rate reductions are seen in the results from the baseline method. For most datasets, the F1 score is highest in ByT5 models, with the exception of the non-preprocessed ICDAR dataset, in which the baseline has the highest F1 score. The F1 scores of the baseline method are not that high on the ICDAR dataset primarily because of low recall.

7.1 Llama results

Although the output of the model was consistently found using regex, the results were not correct for improving character error rate. The model was not fine-tuned, but only told what to do. The Llama model was not able to improve the OCR results for either dataset. The Llama model was not able to consistently output text that resembled the input text, and would often hallucinate extra pieces of output which were not present in the original input text. This leads to a reduction in the text quality, as the original text has not been preserved. Examples of such errors in the output of Llama are given in Section A.

7.2 Custom Dataset

The ByT5 model was fine-tuned on the custom dataset as described in section 6. The results for the different variations of the models and the different preprocessing techniques are given in the top section of Tables 2, 3, 4 and 5.

The leftmost section of each table shows the metrics for the baseline method. The middle section shows the results for the base variant of the ByT5 model, this model is larger than the small model shown on the right side of the figure.

The best results on the custom dataset are seen in the experiment with all preprocessing applied by the fine-tuned ByT5 model with a context length of 50. In this configuration, the CER is reduced by 56%. The best performance of the baseline method is seen in the custom dataset with strange characters removed, with a CER reduction of 48%.

7.2.1 Baseline Methodology

The baseline method improves the CER of the custom dataset in all preprocessing scenarios. The biggest improvement is seen for the dataset with strange characters removed, with 48.1% CER reduction. This is slightly better than for the dataset without preprocessing, where the baseline method gets a 47.8% CER reduction. Interestingly, lowercasing the dataset reduced the performance of this model by a lot: it only reduced the CER by 42% on that dataset. The worst performance of the baseline model was seen on the dataset with all preprocessing techniques applied, in which case it reduced the CER by 35%. Not all preprocessing techniques are thus effective for improving the baselines CER reduction.

The highest F1 score was seen in the experiments with the base dataset without preprocessing

and with the dataset with strange characters removed. This corresponds with the results that the CER is reduced most for these datasets as well. Precision values were very high, above 86% for the lowercased dataset, up to 99% for the dataset with strange characters removed. Recall values were around 50% for all datasets except the one with all preprocessing options enabled, where the recall was only 39%. Precision values were higher than the ByT5 models, but recall values were lower for all datasets. The very low recall values compared to the precision values is also why the F1 scores for the baseline method consistently are lower than those of the ByT5 models.

7.2.2 ByT5 Fine-tuned Model

The ByT5 model was fine-tuned and evaluated on each dataset with context lenghts of 10, 25, 50 or 100 characters. The context length selected made for large differences in CER reduction seen in all datasets. The models with 10 and 100 character context lengths consistently under-performed for both model sizes (small and base), as well as for all datasets.

The best results were seen in the experiments with context lengths of either 25 or 50 characters. The highest overall CER reduction in the thesis was seen in the experiment with all preprocessing techniques enabled, using the ByT5 model with a context length of 50 characters. This yielded a CER reduction of 56%.

The overall best result was seen in the larger base model size. The small model was able to outperform the base model in the experiment with strange character removal and a context length of 25 characters. The results of the base model were more consistent across preprocessing techniques and context lengths however.

The highest F1 scores were seen for the ByT5 models. The base ByT5 model scored the highest overall F1 score of the results in this thesis, at 75% for the all preprocessing custom dataset with a context length of 50 characters. The 50 characters context length base model scored the highest F1 scores in all datasets except for the no strange characters dataset, where the ByT5 small 25-character context length model scored best. The ByT5 models scored better F1 scores in all datasets than the baseline method.

The main reason that F1 scores are higher in the ByT5 models is because of higher recall values. Recall is consistently above 60% for ByT5 base with a context length of 50 characters, compared to the baseline at 50%. Precision scores were not as high for the ByT5 model as for the baseline, with the best scoring F1 models for ByT5 having precision scores between 73% and 85%, whereas the baseline has a precision rate between 86% and 99%.

7.3 ICDAR Dataset

Also for the ICDAR dataset, the different preprocessing techniques have been applied in training and testing of the baseline and ByT5 models. The results for the different variations of the models and the different preprocessing techniques are given in the bottom section of Tables 2, 3, 4 and 5.

The leftmost section of the table shows the metrics for the baseline method. The middle section shows the results for the base variant of the ByT5 model. The results for the small variant of the ByT5 model are shown on the right side of the table.

The best results on the ICDAR dataset are seen in the experiment with the non-preprocessed dataset by the baseline method. It reduces the CER by 21%. the best results on the ICDAR dataset obtained by fine-tuning the ByT5 model is seen in the experiment with all preprocessing, a small context length of 10 characters and the small ByT5 model. It reduces the character error rate by 4% in that experiment.

7.3.1 Baseline Methodology

The baseline method scored worse on the ICDAR datasets than on the custom datasets for all preprocessing options. Still, the method was able to reduce CER in the base dataset by 21%. The method performed worst on the lowercased dataset, only reducing CER by 9%.

F1 scores were also lower for the ICDAR dataset compared to the custom dataset in each of the preprocessing options. The highest F1 score on ICDAR was seen in the experiment of the baseline method on the base ICDAR dataset. For the other datasets, ByT5 models scored better in F1, because of their higher recall values.

Precision of the baseline method on ICDAR datasets is very high, 100% on all datasets except for the lowercased dataset where it scored 73%. Recall is very low for this method however, with the best recall seen in the base dataset at 21%. The lower recall values seen in the preprocessed datasets are the reason that the F1 score for those

falls below the F1 score for the ByT5 models.

7.3.2 ByT5 Fine-tuned Model

The ByT5 models were not able to achieve high CER reduction percentages in any of the context length configurations. The best CER reduction from ByT5 models on the ICDAR datasets was seen in the experiment using ByT5 with a context length of 10 characters on the ICDAR dataset with all preprocessing. Most other experiments yielded negative CER reductions, meaning that CER actually went up after using the model.

Both recall and precision values are very low for the ByT5 models on the ICDAR datasets. The highest recall was 39% for ByT5 small on the lowercased dataset. The highest precision value was seen in the experiment using ByT5 small on the all-preprocessing dataset with a context length of 10 characters. Low precision in the models leads to many mistakes being introduced which increases the CER more than it is reduced.

F1 values are higher for ByT5 models than the baseline models in most ICDAR datasets, even in experiments where CER actually increased. This is because of the low recall that the baseline achieved on the ICDAR data. Still, as the precision of the baseline on ICDAR was very high, not many new errors were introduced while consistently removing a small subsection of the errors, overall leading to a better result than that seen by ByT5 models.

8 Discussion

This thesis has shown the effectiveness of using large language models in correcting OCRed text in a sequence-to-sequence translation setting. The proposed fine-tuned ByT5 base model showed better performance than the currently available baseline OCR correction method tested in our custom modern documents datasets using preprocessing. Llama 0-shot and few-shot learning proved ineffective in improving OCR results mostly due to hallucinations of the model after several tokens of input correction. The ByT5 fine-tuning approach proved ineffective in correcting the publicly available ICDAR dataset. Most configurations of the models showed an increase in the CER on most datasets.

The ICDAR dataset is comprised of 18th, 19th and early 20th century documents, which is likely why the models fail to correct these OCR inputs. The ByT5 model was trained on modern publicly available data with the intention of use in modern products, making it difficult for the model to learn to correct text that looks incorrect in the eyes of a modern English speaker. The baseline model was able to show improvement on this dataset, it does not rely on pre-trained language understanding from modern documents and can thus adapt to the old-English style text without issue. Finetuning of large language models is aimed at modifying the behavior of the model slightly in order to become good at a particular task of interest. This is done after pre-training so that the model already has the basic understanding of the language before learning the specific task that the end-user requires it for [13, 38]. For most applications, this process works very well but for the case of OCR correction on historic documents, the pre-trained models proved incapable of adapting to the new language. The same issue is seen in other tasks that pre-trained LLMs are used on. For example, where the task of named entity recognition is performed with great accuracy on modern texts [42], fine-tuning a named entity recognition LLM for historic documents gives worse results [36]. Pre-training an LLM on historic documents so that it can specifically understand the language used in the target texts can overcome this problem, but requires a very large amount of historic documents. This task has been performed successfully by Schweter et al. [36], who pre-trained a custom BERT model using historic documents, after which they fine-tuned it on the named entity recognition task to get competitive results in the HIPE-2022 competition.

For the custom dataset, the ByT5 base model with a context length of 50 characters performed the best out of all models. On the dataset with all preprocessing options, 56% of character errors found in the text before correction were removed. This is a large improvement compared to the baseline method which was only able to achieve a 48% reduction in CER in the best case on the dataset that had removed strange characters.

The results show that in combination with the LLM models, preprocessing is able to increase the performance of the models a lot. The best result achieved without preprocessing is 8 percentage points worse in removing OCR errors than the best result achieved with preprocessing. Not all preprocessing options are equally effective, and the effectiveness is dependent on the model deployed. For the ByT5 models, lowercasing the input raised the effectiveness of error correction most, whereas this was the second worst option

Model	Baseline	ByT5-base				ByT5-small			
Context Length	30	10	25	50	100	10	25	50	100
Dataset									
custom all preprocessing	0.3503	0.5041	0.5379	**0.5564**	0.4665	0.4944	0.5277	0.4744	0.3953
custom base dataset	**0.4780**	0.4139	0.4560	0.4178	0.3414	0.4119	0.4585	0.3747	0.3496
custom lowercased	0.4185	0.3975	0.4510	**0.4770**	0.2999	0.4194	0.4442	0.4457	0.2843
custom no strange characters	0.4813	0.4392	0.4773	0.3877	0.4061	0.4533	**0.5030**	0.4494	0.2705
ICDAR all preprocessing	**0.1765**	0.0185	-0.1350	-0.2613	-2.0097	0.0398	-0.0899	-2.0226	-1.2945
ICDAR base dataset	**0.2083**	-0.1789	-0.1566	-0.4590	-1.1584	0.0239	-0.3140	-0.3574	-0.8392
ICDAR lowercased	**0.0929**	-0.1570	-0.1285	-0.3125	-1.6214	-0.1224	-0.3385	-0.8723	-2.0093
ICDAR no strange characters	**0.1176**	-0.0186	-0.1297	-0.3355	-2.6247	-0.0238	-0.1814	-1.1110	-1.1123

Table 2: CER reduction of the models on the different datasets as a fraction of pre-OCR correction error rate

Model	Baseline	ByT5-base				ByT5-small			
Context Length	30	10	25	50	100	10	25	50	100
Dataset									
custom all preprocessing	0.5455	0.6980	0.7357	**0.7512**	0.7228	0.6885	0.7235	0.7196	0.6907
custom base dataset	0.6581	0.6459	0.6886	**0.6914**	0.6633	0.6419	0.6858	0.6733	0.6567
custom lowercased	0.6313	0.6306	0.6863	**0.7092**	0.6591	0.6376	0.6760	0.6885	0.6460
custom no strange characters	0.6527	0.6632	0.7042	0.6848	0.6954	0.6616	**0.7086**	0.6949	0.6453
ICDAR all preprocessing	0.3000	0.2483	0.2771	**0.3056**	0.1835	0.2269	0.2922	0.1885	0.2202
ICDAR base dataset	**0.3448**	0.2475	0.3009	0.2489	0.2241	0.2690	0.2817	0.2910	0.2216
ICDAR lowercased	0.2462	0.2785	**0.3028**	0.2956	0.1943	0.2709	0.2817	0.2407	0.2046
ICDAR no strange characters	0.2105	0.2515	**0.2752**	0.2679	0.1624	0.2433	0.2601	0.2304	0.1953

Table 3: F1 scores of the models on the different datasets

for preprocessing in the baseline test. Lowercasing is effective in reducing the number of possible characters the model sees. This can be helpful for fine-tuning the ByT5 model which now only has to focus on the content of the text without needing to perform on predicting the case of the text, allowing it to more quickly adapt to the task of OCR post-correction. Lowercasing also removes some information however. Capital letters are used in language for pre-determined reasons, giving extra information on the words that can be expected. As also noted by Brandsen et al. in [3], when preprocessing the input data, the right balance needs to be struck between reducing the variation of the input so that the models are able to learn the patterns we care about, whilst not throwing out too much information which may be helpful in finding the patters. Potentially, the baseline method which did not have any pretraining done was able to effectively use case-information in the correction decisions, leading to a drop in performance when such information was not available.

For the ICDAR datasets, preprocessing did help improve scores in the best experiments for the ByT5 models, but for the baseline method preprocessing only hindered CER reduction. This shows that before selecting pre-processing techniques, the effectiveness of such techniques for the specific model deployed need to be evaluated, as not all models work well with preprocessing.

One- and few-shot learning showed not to be an effective method for enabling OCR post-correction in Llama models. The models did not consistently output text based on the input given, and often hallucinate large portions of text. This leads to large mismatches between the output text and the ground-truth text, as the ground-truth text in most cases largely resembles the input text.

8.1 Limitations

In this thesis the general feasibility of using large language models in OCR post-correction is shown. There are also some limitations however. For one, it is not shown that the methodology proposed here is functional on datasets that are publicly available. In searching for a dataset to test this on, it proved difficult to find one that is representative of the task for which this methodology was designed. It is

Model	Baseline	ByT5-base				ByT5-small			
Context Length	30	10	25	50	100	10	25	50	100
Dataset									
custom all preprocessing	0.3898	0.5730	0.6430	0.6696	**0.6956**	0.5586	0.6178	0.6743	0.6751
custom base dataset	0.5024	0.5346	0.6016	0.6523	**0.6487**	0.5270	0.5910	0.6444	0.6221
custom lowercased	0.4978	0.5143	0.6005	0.6378	**0.6768**	0.5106	0.5799	0.6126	0.6530
custom no strange characters	0.4875	0.5521	0.6222	0.6652	**0.6779**	0.5344	0.6044	0.6269	0.6637
ICDAR all preprocessing	0.1765	0.1621	0.2176	0.2776	0.3383	0.1409	0.2250	**0.3509**	0.3240
ICDAR base dataset	0.2083	0.1939	0.2490	0.2417	**0.3116**	0.1796	0.2577	0.2785	0.2618
ICDAR lowercased	0.1481	0.2233	0.2451	0.2754	0.3162	0.2085	0.2625	0.2968	**0.3870**
ICDAR no strange characters	0.1176	0.1711	0.2145	0.2444	**0.3514**	0.1646	0.2077	0.3160	0.2563

Table 4: Recall scores of the models on the different datasets

Model	Baseline	ByT5-base				ByT5-small			
Context Length	30	10	25	50	100	10	25	50	100
Dataset									
custom all preprocessing	**0.9079**	0.8927	0.8595	0.8554	0.7522	0.8969	0.8727	0.7713	0.7070
custom base dataset	**0.9537**	0.8158	0.8051	0.7356	0.6786	0.8206	0.8169	0.7050	0.6954
custom lowercased	**0.8626**	0.8150	0.8006	0.7986	0.6423	0.8484	0.8104	0.7859	0.6391
custom no strange characters	**0.9873**	0.8302	0.8111	0.7056	0.7138	0.8682	0.8563	0.7793	0.6280
ICDAR all preprocessing	**1.0000**	0.5302	0.3816	0.3400	0.1259	0.5823	0.4167	0.1288	0.1668
ICDAR base dataset	**1.0000**	0.3421	0.3804	0.2565	0.1749	0.5356	0.3107	0.3046	0.1921
ICDAR lowercased	**0.7283**	0.3700	0.3961	0.3190	0.1403	0.3866	0.3040	0.2025	0.1391
ICDAR no strange characters	**1.0000**	0.4743	0.3839	0.2965	0.1056	0.4663	0.3480	0.1813	0.1577

Table 5: Precision scores of the models on the different datasets

important that the documents have character error rates low enough for the model to get a general understanding of the task, but datasets of modern documents in particular with low character error rates were not available.

Another limitation of this thesis is that it was not possible to test the large variant of the ByT5 model for OCR correction. This is due to a hardware constraint. Fine-tuning large language models takes vast resources and time, in particular the amount of VRAM on available GPU devices is leading in the feasibility of fine-tuning LLMs. This thesis showed that the larger base ByT5 model performed much better than the small ByT5 model, indicating that a larger variant of ByT5 might see more improvement.

8.2 Future Work

For an improvement on the work done here, future research could validate if the 50% CER improvement found here is also achievable in other modern document datasets. The best way to investigate this would be by organizing a new competition specifically for modern documents. This requires curating the ground-truth for a set of modern documents, to create a dataset on which future methodologies can be tested.

Another question left open is whether larger language models perform better at the OCR post-correcting task than the ByT5 model sizes tested here. Future work investigating the effectiveness of larger language models should make use of several GPUs with enough VRAM to train these larger models. For additional speedup, the different experiments like they are done in this thesis can be performed in parallel when more GPUs are available.

For future work, researchers could investigate the possibility of fine-tuning the open-source Llama models for OCR post-correction. The Llama models have substantially larger architectures than the ByT5 model fine-tuned in this thesis, making it challenging to get the right resources to be able to do so effectively, but the models also boast stronger results in benchmark tests. Fine-tuning Llama models on the OCR post-correction task may reduce issues with hallucination which would fix the main problem encountered with the model here. Wu et

al. [47] describe the process of finetuning a Llama model for question-answering on a domain specific task. Their published code can be adapted to finetune the model for OCR post-correcting.

Future research could also investigate the possibility of pre-training a large language model on older texts specifically, leading to a model which might be more receptive to training to correct OCR errors in this niche of documents in particular. In that way, the effectiveness of large language models for OCR correction on old texts can be properly evaluated. This was done for the task of named entity recognition in [36]. The same methodology can be adapted to fine-tune ByT5 for historic document OCR correction.

As recall showed to be one of the areas which could use more improvement in the ByT5 models fine-tuned here, a hybrid approach making use of an OCR-error detector alongside the ByT5 model for OCR-error correction could be developed. Such an OCR-error detector is part of the two-step approach found in Schaefer et al. [35], Using a similar two-step approach could show improvements above the methodology here, as the model could be more certain of the need for character modification when the helper algorithm indicates possible OCR errors.

An optimisation that can be done to slightly improve the results found here is to investigate the exact context length that balances sufficient context without overwhelming the model with input. In this thesis it is shown that too little context leads to worse performance, but too much context also reduces performance of the model. Within the range between 10 to 50 characters this thesis has tested the context length of 25 characters, but the ideal context length could be anywhere in this range. For a final optimization of the methodology, the optimal context length could be investigated. Through a sensitivity analysis where the context length is increased by 1 character each time, the optimal context length can be found which could slightly improve on the results seen here.

The size of the model is more important than the context length. A larger version of ByT5 likely scores significantly better at the OCR post-correction task. This thesis shows the feasibility of using LLMs for OCR post-correction, but future work could improve on this by deploying larger more resource-intensive models for the best performance. Depending on the need for super high-quality OCR output, the added compute time cost of implementing these more resource heavy models might be worth it, but this has to be analyzed on a case-by-case basis, as in most cases the slight benefit of OCR error correction over simpler models might not be worth the extra latency and processing cost that would come with deploying the larger models.

9 Conclusion

9.1 Answers to Research questions

The research questions were introduced in section 1.6. Here I will shortly answer each of the research questions based on the results of the experiments.

- **RQ1:** Can character-level LLMs perform better in correcting OCR output than state-of-the-art non-pretrained language model based methods?
  - The character-level LLM ByT5 outperforms the baseline of a state-of-the-art non-pretrained language model on modern documents. However, it does not perform better on the ICDAR dataset.
- **RQ2:** Do finetuned character-level LLMs outperform the generative LLM Llama in post-correcting OCRed text?
  - Yes, finetuning the character-level LLM ByT5 results in better performance than the generative LLM Llama. While Llama struggles to reliably correct OCR errors, a finetuned version of ByT5 is effective in reducing these errors.
- **RQ3:** How do preprocessing techniques impact the effectiveness of the LLM post-OCR correcting techniques deployed?
  - Preprocessing has a large impact on the effectiveness of ByT5 for error correction. Techniques such as lowercasing and removing strange characters greatly enhance the model's ability to reduce the character error rate.

9.2 ByT5 for OCR post-correction

The ByT5 models fine-tuned in this thesis show the capability of recognizing and correcting OCR errors in modern documents. The ability to do so is strongly dependent on the context length given to the models and the size of the models. The larger "base" ByT5 model showed capability in learning to correct OCR errors with the best configuration of

the input length of 50 characters on the dataset with all preprocessing options, reducing the number of OCR errors by 56%. This performance was better than the performance of the baseline model which scored a CER reduction of 48% on the dataset with strange characters removed. The precision of the baseline model was higher for all datasets, but the recall was worse, leading to a lower level of CER reduction for the baseline. This is also represented in the higher F1 scores for the ByT5 models compared to the baseline model in all datasets.

Context length has a strong impact on the effectiveness of the ByT5 models. differences in CER reduction of 13 percentage points are seen between the best and worst selection of context lengths. In the case of ByT5 small on the all preprocessing dataset, choosing a context length of 25 characters reduced CER by 53%, whereas using the same model but with a context length of 100 characters reduced CER by only 40%. This shows that context length is a very important parameter to get right when fine-tuning pre-trained large language models on OCR post-correction. This is likely due to the model having trouble "keeping in line" as the context length increases, with the model tending towards more random generation as opposed to the specific task of correcting the input text.

For the best performing configurations of the base model, the recall was much worse than the precision, indicating that the models are generally able to leave correct OCR text as-is, whilst sometimes being able to correct encountered OCR errors. In the outputs of the models with low precision it means that a lot of characters which were correct in the OCR output were modified for the prediction. This modification then leads to an error. As there are far more correct than incorrect characters in the original input, modifying characters when not absolutely sure that they are in fact incorrect leads to the introduction to many new errors. It is thus more important that a model has high precision than high recall. Missing a few errors (low recall) is less bad than incorrectly adding errors through low precision.

The highest recall was 0.696 from the 100-character context length base model on the dataset with all preprocessing options. This shows that the area where most improvement is still to be had is in the recall ability of the models. The models are often able to correctly identify if a piece of text is correct and should not be modified, but tend to conserve OCR errors in order to prevent making false-positive modifications.

Although ByT5 models have shown slightly better performance on the custom dataset, they were not able to improve the OCR on the ICDAR dataset. Besides the higher performance on the custom dataset, the increased processing time due to the larger model size compared to the baseline makes it hard to justify using the model. Only in settings where the output needs to have the best possible result, would using ByT5 be preferable over the simpler baseline solution.

9.3 Llama for OCR post-correction

The Llama model did not show promising results in correcting OCR text. zero-shot, as well as few-shot learning did not get the model to perform the task of correcting OCR errors reliably. In some occasions the original text was left mostly intact, with some corrections made, but in most cases only the beginning of the original text was kept, where the rest of the output consisted of hallucinated text that could have followed the beginning of the input text but which was not in fact in the true input. This leads to a large mismatch between the predicted text and the ground-truth text, making this method of OCR correcting infeasible for practical use in document error correction.

A Example Outputs of the Models

Example Outputs on ICDAR Dataset

ByT5

Semi-Correct Output. Some of the characters in this example are correctly modified. "thert" was correctly changed to there, and "theit" to "their". It did not correctly identify that "we<" should have become "was", and the "," has incorrectly been modified to be a ";".

```
INPUT  : thert we<, theit pots a
GT     : there was, their pots a
OUTPUT : there we ; their pots a
```

Baseline

Correct Output. The baseline method correctly modified the word offenffve to offensive. It also started the quote that comes after it.

```
INPUT  : which are offenffve : Not to l
GT     : which are offensive:' Not to l
OUTPUT : which are offensive:' Not to l
```

Example Outputs on Custom Dataset

ByT5

Correct Output. The model has correctly identified that the comma in front of "burgemeester" should not have been there. It also notes that a space is likely to come after the word "burgemeester". It did not correctly predict the last character of the window, but this is to be expected as this could have realistically been a variety of characters.

```
INPUT  : lege van ‘burgemeester@@
GT     : lege van @burgemeester e
OUTPUT : lege van @burgemeester v
```

Wrong Output. In this example, the word "ruimtelijkfysieke" has many mistakes in the OCR text. Because of these mistakes, the model incorrectly assumes that the correct word should have been "ruimtelijke". It therefore omits a large section of the word and adds more mistakes than corrections.

```
INPUT  : gie op het hele ruimteli@xtysieke domein. voor dit
GT     : gie op het hele ruimtelijkfysieke domein. voor dit
OUTPUT : gie op het hele ruimtelij@@@@@@ke domein. voor dit
```

Long Input Hallucination. In this example, the model is given a long input string. After the first few characters, the prediction becomes surrealistic, as the output shows a continuous sequence of alterations on the word "lokalings". A lot of mistakes are added in this output.

```
INPUT  : @ings@sbiodn die voorkomen in diverse landschapsty pen
                en waar eer ruimte is voor antwikkeling@@ van
GT     : lingsgebieden die voorkomen in diverse landschapstypen
                en waa@@@@r@ruimte is voor ontwikkeling@@ van
OUTPUT : lings@@@@lokalingslo@k@@@alin@@@gslokalings@@@@lokalin@
                gsloka@lingslokalings@lokalingslokalingslokal
```

Baseline

Correct Output. In this example, the model correctly identifies that "_" should have been omitted.

```
INPUT  : e planregels, veranderd _daadwe
GT     : e planregels, veranderd @daadwe
OUTPUT : e planregels, veranderd @daadwe
```

Missed Identification Of Mistake. Here the baseline incorrectly modifies the "-" from the input into a "·". In the GT text this should have been omitted entirely, leaving only a space.

```
INPUT  : uurzame woonomgeving, - in rel
GT     : uurzame woonomgeving, @@in rel
OUTPUT : uurzame woonomgeving, · in rel
```

Llama errors

Llama hallucinates text to the end of the correction. This extra text leads to very high error rates, as all the extra characters are incorrect.

```
INPUT  : waarbi) : aanpak van de bron prevaleert boven maatregelen
GT     : waarbi) : aanpak van de bron prevaleert boven maatregelen
OUTPUT : waarbi) : aanpak van de bron prevaleert boven maatregelen en de afwenteling van lokale
```

Llama misses part of the output. Here the Llama model fails to include the last bit of text in its output, leading to a large portion of text being missing which results in large character error rates.

```
INPUT  : water van _ verschillende kwaliteit wordt zoveel mogelik gescheiden. ‘wanneer ook berg
GT     : water van@@ verschillende kwaliteit wordt zoveel mogelijk gescheiden. @wanneer ook berg
OUTPUT : water van verschillende kwaliteit wordt zoveel mogelik gescheiden.
```

References

[1] Toufique Ahmed and Premkumar Devanbu. "Few-shot training LLMs for project-specific code-summarization". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–5.

[2] Dimitris Alikaniotis and Vipul Raheja. "The Unreasonable Effectiveness of Transformer Language Models in Grammatical Error Correction". In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Ed. by Helen Yannakoudakis et al. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 127–133. DOI: 10.18653/v1/W19-4412. URL: https://aclanthology.org/W19-4412.

[3] Alex Brandsen. "Information extraction and machine learning for archaeological texts". In: *Discourse and Argumentation in Archaeology: Conceptual and Computational Approaches*. Springer, 2023, pp. 229–261.

[4] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[5] Guillaume Chiron et al. "ICDAR2017 competition on post-OCR text correction". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 1. IEEE. 2017, pp. 1423–1428.

[6] Jacob Devlin et al. "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)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[7] Xiaoqing Ding, Li Chen, and Tao Wu. "Character independent font recognition on a single chinese character". In: *IEEE Transactions on pattern analysis and machine intelligence* 29.2 (2007), pp. 195–204.

[8] John Evershed and Kent Fitch. "Correcting noisy OCR: Context beats confusion". In: *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. 2014, pp. 45–51.

[9] Karez Hamad and Kaya Mehmet. "A detailed analysis of optical character recognition technology". In: *International Journal of Applied Mathematics Electronics and Computers* Special Issue-1 (2016), pp. 244–249.

[10] Pengcheng He, Jianfeng Gao, and Weizhu Chen. "DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=sE7-XhLxHA.

[11] Pengcheng He et al. "{DEBERTA}: {DECODING}-{ENHANCED} {BERT} {WITH} {DISENTANGLED} {ATTENTION}". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=XPZIaotutsD.

[12] Thomas Hegghammer. "OCR with Tesseract, Amazon Textract, and Google Document AI: a benchmarking experiment". In: *Journal of Computational Social Science* 5.1 (2022), pp. 861–882.

[13] Wenbo Hu et al. "Bliva: A simple multimodal llm for better handling of text-rich visual questions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 3. 2024, pp. 2256–2264.

[14] Abir Jaafar Hussain, Ali Al-Fayadh, and Naeem Radi. "Image compression techniques: A survey in lossless and lossy algorithms". In: *Neurocomputing* 300 (2018), pp. 44–69.

[15] Vinh-Nam Huynh, Ahmed Hamdi, and Antoine Doucet. "When to use OCR post-correction for named entity recognition?" In: *Digital Libraries at Times of Massive Societal Transition: 22nd International Conference on Asia-Pacific Digital Libraries, ICADL 2020, Kyoto, Japan, November 30–December 1, 2020, Proceedings 22*. Springer. 2020, pp. 33–42.

[16] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *CoRR* abs/2001.08361 (2020). arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361.

[17] Jan Kocoń et al. “ChatGPT: Jack of all trades, master of none”. In: *Information Fusion* (2023), p. 101861.

[18] Karen Kukich. “Techniques for automatically correcting words in text”. In: *ACM computing surveys (CSUR)* 24.4 (1992), pp. 377–439.

[19] Alex Kuznetsov and Hector Urdiales. “Spelling Correction with Denoising Transformer”. In: *CoRR* abs/2105.05977 (2021). arXiv: 2105.05977. URL: https://arxiv.org/abs/2105.05977.

[20] Kalev Leetaru. “Mass book digitization: The deeper story of Google Books and the Open Content Alliance”. In: *First Monday* (2008).

[21] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[22] William B Lund, Daniel D Walker, and Eric K Ringger. “Progressive alignment and discriminative error correction for multiple OCR engines”. In: *2011 International Conference on Document Analysis and Recognition*. IEEE. 2011, pp. 764–768.

[23] Lijun Lyu et al. “Neural OCR post-hoc correction of historical corpora”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 479–493.

[24] Sabrina J Mielke et al. “Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP”. In: *arXiv preprint arXiv:2112.10508* (2021).

[25] David Miller et al. “Named entity extraction from noisy input: speech and OCR”. In: *Sixth Applied Natural Language Processing Conference*. 2000, pp. 316–324.

[26] Netherlands Ministry of Infrastructure and Water Management. *Ruimtelijkeplannen.nl*. Accessed 2024. URL: https://www.ruimtelijkeplannen.nl/.

[27] Thi Tuyet Hai Nguyen et al. “Neural machine translation with BERT for post-OCR error detection and correction”. In: *Proceedings of the ACM/IEEE joint conference on digital libraries in 2020*. 2020, pp. 333–336.

[28] Thi Tuyet Hai Nguyen et al. “Survey of post-OCR processing approaches”. In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–37.

[29] Kostiantyn Omelianchuk et al. “GECToR – Grammatical Error Correction: Tag, Not Rewrite”. In: *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Ed. by Jill Burstein et al. Seattle, WA, USA → Online: Association for Computational Linguistics, July 2020, pp. 163–170. DOI: 10.18653/v1/2020.bea-1.16. URL: https://aclanthology.org/2020.bea-1.16.

[30] Juan Antonio Ramirez-Orta et al. “Post-ocr document correction with large ensembles of character sequence-to-sequence models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 10. 2022, pp. 11192–11199.

[31] Martin Reynaert. “OCR post-correction evaluation of early dutch books online-revisited”. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*. 2016, pp. 967–974.

[32] Christophe Rigaud et al. “ICDAR 2019 Competition on Post-OCR Text Correction”. In: *Proceedings of the 15th International Conference on Document Analysis and Recognition (2019)*. 2019.

[33] Christophe Rigaud et al. “ICDAR 2019 competition on post-OCR text correction”. In: *2019 international conference on document analysis and recognition (ICDAR)*. IEEE. 2019, pp. 1588–1593.

[34] Michael S Rosenberg. *Sequence alignment: methods, models, concepts, and strategies*. Univ of California Press, 2009.

[35] Robin Schaefer and Clemens Neudecker. “A two-step approach for automatic OCR post-correction”. In: *Proceedings of the The 4th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*. 2020, pp. 52–57.

[36] Stefan Schweter et al. “hmbert: Historical multilingual language models for named entity recognition”. In: *arXiv preprint arXiv:2205.15575* (2022).

[37] Lukas Stankevičius et al. “Correcting diacritics and typos with a ByT5 transformer model”. In: *Applied Sciences* 12.5 (2022), p. 2636.

[38] Riste Stojanov et al. “A fine-tuned bidirectional encoder representations from transformers model for food named-entity recognition: Algorithm development and validation”. In: *Journal of Medical Internet Research* 23.8 (2021), e28229.

[39] Gustavo Sutter Pessurno de Carvalho. “Multilingual Grammatical Error Detection And Its Applications to Prompt-Based Correction”. MA thesis. University of Waterloo, 2024.

[40] Tesseract OCR. *Tesseract OCR 4.1.1*. GitHub Repository. https://github.com/tesseract-ocr/tesseract. 2019.

[41] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971.

[42] Asahi Ushio and Jose Camacho-Collados. “T-NER: an all-round python library for transformer-based named entity recognition”. In: *arXiv preprint arXiv:2209.12616* (2022).

[43] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[44] Martin Volk, Lenz Furrer, and Rico Sennrich. “Strategies for reducing and correcting OCR errors”. In: *Language Technology for Cultural Heritage: Selected Papers from the LaTeCH Workshop Series*. Springer. 2011, pp. 3–22.

[45] Luis Von Ahn et al. “recaptcha: Human-based character recognition via web security measures”. In: *Science* 321.5895 (2008), pp. 1465–1468.

[46] Jules White et al. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. 2023. arXiv: 2302.11382.

[47] Chaoyi Wu et al. “Pmc-llama: Further finetuning llama on medical papers”. In: *arXiv preprint arXiv:2304.14454* (2023).

[48] Linting Xue et al. “Byt5: Towards a token-free future with pre-trained byte-to-byte models”. In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 291–306.

[49] Yazhou Zhang et al. *DialogueLLM: Context and Emotion Knowledge-Tuned Large Language Models for Emotion Recognition in Conversations*. 2024. arXiv: 2310.11374.