Tokenization Repair in the Presence of Spelling Errors

Hannah Bast, Matthias Hertel, Mostafa M. Mohamed
University of Freiburg
Freiburg, Germany
{bast,hertelm,amin}@cs.uni-freiburg.de

Abstract

We consider the following *tokenization repair problem*: Given a natural language text with any combination of missing or spurious spaces, correct these. Spelling errors can be present, but it's not part of the problem to correct them. For example, given: "*Tispa per isabout token izaionrep air*", compute "*Tis paper is about tokenizaion repair*".

We identify three key ingredients of high-quality tokenization repair, all missing from previous work: deep language models with a bidirectional component, training the models on text with spelling errors, and making use of the space information already present. Our methods also improve existing spell checkers by fixing not only more tokenization errors but also more spelling errors: once it is clear which characters form a word, it is much easier for them to figure out the correct word.

We provide six benchmarks that cover three use cases (OCR errors, text extraction from PDF, human errors) and the cases of partially correct space information and all spaces missing. We evaluate our methods against the best existing methods and a non-trivial baseline. We provide full reproducibility under `https://ad.cs.uni-freiburg.de/publications`.

1 Introduction

Tokenizing a given text into words is the first step in many natural language processing applications, including: search engines, translation services, spell checkers and all kinds of learning tasks performed on text. This tokenization is typically performed by the following simple method or a variant of it: define a set of word characters and take each maximal sequence of word characters as one token. For example, for

This algoritm runs in linear time (1)

a simple such tokenization yields the six words

This, algoritm, runs, in, linear, time.

Note the spelling error in the second word. Spelling correction is not part of tokenization. We come back to this important aspect in Section 1.2.

Missing and spurious spaces are common errors in digital text documents. We refer to the union of both types of errors as *tokenization errors*. Here is a variant of the sentence above with one missing space and one spurious space:

This algor itm runsin linear time (2)

In this paper, we consider the following *tokenization repair* problem: Given a sequence of characters representing a natural language text, with an arbitrary amount of missing and spurious spaces and possibly also with spelling errors, compute the variant of the text with correct spacing. For example, given (2) above, compute (1).

Tokenization repair can be considered as a generalization of the *word segmentation* problem, where the text is given without any space information. Indeed, we also evaluate our methods on this special case in Section 5.

1.1 Sources of tokenization errors

Tokenization errors are typical in texts that are digitized by Optical Character Recognition (OCR) techniques. For example, tokenization errors are known to be frequent in the ACL anthology corpus (Nastase and Hitschler, 2018) and in digitized newspapers (Soni et al., 2019; Adesam et al., 2019). Many OCR error correction methods can not deal with tokenization errors, and it is stated in Hämäläinen and Hengchen (2019) that:

"A limitation of our approach is that it cannot do word segmentation in case multiple words have been merged together as a result of the OCR process. However, this problem is complex enough on its own right to deserve an entire publication of its own and is thus not in the scope of our paper."

Portable document formats like PDF specify the position of the characters on a page. When extracting text from such formats, space positions must be inferred from the distance between the characters' bounding boxes, which is error-prone. Tokenization errors can also be found in human-typed texts. The fraction of these errors among misspellings was found to be 15 % in Kukich (1992).

Tokenization errors degrade the performance of any natural language processing (NLP) system, if it does not account for them. A search engine will not find "*algorithm*" in a document containing "*algo rithm*". Syntax parsers and word labelers will not give the correct results if a word is split into multiple tokens, or multiple words merged into one. A text classifier based on word statistics or word vector representations will fail to retrieve statistics or vector representations for wrongly tokenized words, which can result in wrong classifications.

1.2 Tokenization and Spelling Correction

Existing spelling correction tools either assume correct tokenization or fix tokenization errors only to a limited extent; see our evaluation in Section 5. We show that by using our tokenization repair as a pre-processing, not only are more tokenization errors fixed (obviously), but also more spelling errors. The reason is that once it is clear which characters form a word, it is much easier to figure out the correct word. For example, the best spelling corrector from our evaluation corrects the passage "*Mqr ymay have k i ssecl John*" from the ACL anthology to "*Mqr may have k i secl John*" without prior tokenization repair and to the correct "*Mary may have kissed John*" after tokenization repair.

It seems that, ideally, tokenization and spelling errors should be fixed together. However, this appears to be a very hard problem. Our own approaches can be adapted to also fix spelling errors, but only with an impractical large running time. There is a fundamental reason for this: because of the very many possible interpretations, the size of the *beam* (containing the best partial corrections of the sequence) needs to be very large, in order not to miss the correct solution; see Section 3.1. All other tools we know of that consider both tokenization and spelling errors (and some even grammar errors), fare very poorly on passages with tokenization and spelling errors combined.

1.3 Contributions

We consider these as our main contributions:

• We consider the problem of tokenization repair in the presence of spelling errors. Unlike previous work, our model can make use of the space information already present. We can also solve the classical word segmentation problem (all spaces removed) better. We show that it is crucial to train on text with (the right kind and dose of) spelling errors.

• In previous work, forward models combined with a beam search gave the best results. We present an elegant idea to realize a bidirectional model; see Figure 1 for an illustration. This is tricky for a task like tokenization repair, which involves changing the sequence while correcting it.

• We provide six benchmarks that cover all our use cases (OCR errors, text extraction from PDFs, human errors) and different degrees of available space information (partially correct or all spaces missing). We make use of existing benchmarks wherever possible (manually augmenting some by a ground truth) and create new benchmarks with realistic error models.

• We compare our approach with the best existing methods (from the literature) and tools (commercial and open-source). We evaluate both the quality of the tokenization repair and how existing spelling correctors perform much better when the space errors are fixed first.

• We provide a single model, trained across multiple corpora, that produces good results across all benchmarks without the need for any fine-tuning or hyperparameter optimization.

• Our code, data, benchmarks and trained models are available under `https://ad.cs.uni-freiburg.de/publications`. It includes a Docker setup that allows an easy replication of all our results, a web application that allows an interactive error analysis for all methods and benchmarks, and a version of the ACL anthology corpus where the tokenization errors were corrected by our best method.

2 Related Work

A beam search with neural and character n-gram language models is used for word segmentation in Doval and Gómez-Rodríguez (2019). In Section 5, we evaluate our own implementation of this approach on our benchmark and we also compare against their results on their benchmark. We improve on their approach in several respects: integrating a bidirectional model (which is not trivial), considering the given spaces in the input (they remove all spaces), and explicitly addressing typos by incorporating error models in training (they test their approach on tweets, but do not explicitly handle typos).

Tokenization repair on the ACL anthology corpus is done in Nastase and Hitschler (2018) with a neural machine translation model translating from the sequence without spaces to the sequence with spaces. They also remove all spaces from the input text, thus discarding valuable information. In Section 5.3, we compare our results against theirs.

A beam search with a word bigram language model, instead of a character-based language model, is used in Mikša et al. (2010) to correct missing spaces in Croatian texts that were digitized by OCR. In Soni et al. (2019), n-gram statistics are used to determine when to split an out-of-vocabulary token into two words. By using neural language models, we extend the scope of this context beyond the boundaries of n-gram models.

Recent work on Chinese and Arabic word segmentation uses bidirectional neural network models to predict word boundaries, e.g. based on bi-LSTMs (Ma et al., 2018; Almuhareb et al., 2019) or a pre-trained BERT model (Huang et al., 2020). These models are not directly applicable for our task of tokenization repair for the English language, since our inputs contain spurious and missing spaces and the English BERT model (Devlin et al., 2019) is pre-trained on text with correct tokenization.

There is a large body of research on OCR post-correction (Tong and Evans, 1996; Taghva and Stofsky, 2001; Niklas, 2010; Kissos and Dershowitz, 2016; Chiron et al., 2017; Dong and Smith, 2018; Rigaud et al., 2019; Hämäläinen and Hengchen, 2019; Nguyen et al., 2019, 2020), which usually does not address tokenization errors explicitly. Only few publications provide code so that we can evaluate their tokenization repair capabilities.

3 Approach

Our approach is a beam search based on deep character-based models, unidirectional and bidirectional.

3.1 Character-based models

We represent the strings as sequences of one-hot encoded characters, where we use the 200 most frequent characters, while replacing the others by a special character UNK for unknown characters. Sequences are appended with start and end of sentence special characters (SOS and EOS).

3.1.1 Unidirectional language models

Character-based language models estimate the probability of a string to occur in some language based on the probabilities of the individual characters in the string. Following Graves (2013), we implement these models as recurrent neural networks, using LSTM cells. Our architecture consists of an LSTM cell with 1024 units, followed by a dense layer (with 1024 units and ReLU activation) and a softmax output layer for character classification. This architecture has 6,287,563 trainable parameters, which are trained with the categorical cross entropy loss. The model predicts $\overrightarrow{p}(s|c)$, which is the probability that a character s comes after a context c. Moreover, we define $\overrightarrow{p}(\llcorner s|c) := \overrightarrow{p}(s|c\llcorner) \cdot \overrightarrow{p}(\llcorner|c)$ as the probability that a space and character s come after context c.

3.1.2 Bidirectional sequence labeling model

We utilize a bidirectional model that predicts the probability $\overleftrightarrow{p_i}$ of having a space before the i^{th} character when the whole sequence of non-space characters is the input. Our architecture consists of a bidirectional LSTM cell with 1024 units, followed by a dense layer (with 1024 units and ReLU activation) and a sigmoid output unit. This architecture has 12,158,980 trainable parameters, which are trained with the binary cross entropy loss.

3.2 Beam search

Beam search is a search algorithm similar to breadth-first search, but instead of maintaining all search states at a given level, it maintains only the best b states, which correspond to an estimation of the best b partial solutions (called *beams*) (Medress et al., 1977). We introduce two variants of beam search, unidirectional (UNI) and bidirectional (BID).

Correction procedure: Given a mistokenized string Q, with its corresponding sequence of m non-space characters T, the procedure executes beam search for m levels. At level i, given a partial solution's search state (S_{i-1}, R_{i-1}) of accumulated score and partial solution string, two candidate extensions are created:

1. Adding T_i without space, which results in:
$$S_i = S_{i-1} - \log \overrightarrow{p}(T_i|R_{i-1}) + P_{del}$$
$$R_i = R_{i-1}T_i$$
2. Adding a space before T_i, which results in:
$$S'_i = S_{i-1} - \log \overrightarrow{p}(\llcorner T_i|R_{i-1}) + P_{ins}$$
$$R'_i = R_{i-1}\llcorner T_i$$

P_{del} and P_{ins} are non-negative penalties that are used only when the introduced extension is not originally in Q, otherwise they are equal to 0. In other words, considering the character Q_{j-1} preceding T_i in the input sequence, P_{ins} is used when $Q_{j-1} \neq \text{␣}$ and P_{del} is used when $Q_{j-1} = \text{␣}$. The penalties regularize the effect of making too many edits. These equations correspond to UNI.

The final solution R^* is the estimated corrected sequence of lowest penalized negative log-likelihood score (highest probability):

$$-\log p(R^*) + n_{ins} P_{ins} + n_{del} P_{del}$$

where n_{ins} is the number of space insertions and n_{del} is the number of space deletions, and

$$p(R^*) = \prod_{i=1}^{|R^*|} \overrightarrow{p}(R^*_i | R^*_{1:(i-1)})$$

The time complexity is $\mathcal{O}(|Q| \cdot b)$, because we process $2b$ candidates at m levels ($m \leq |Q|$). We use a beam size $b = 5$ in our implementation. As a result, the algorithm runs in linear time.

Beam search bidirectional (BID): This method combines UNI with the bidirectional labeling model introduced in section 3.1.2. We process the non-space characters T using the bidirectional model, then we modify the beam search formulas:

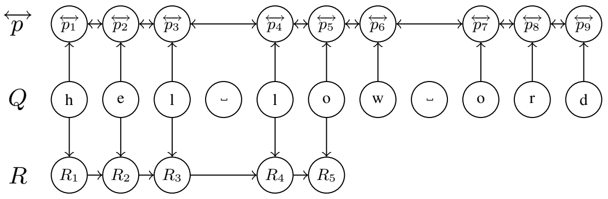
$$S_i = S_{i-1} - \log(\overrightarrow{p}(T_i | R_{i-1}) \cdot (1 - \overleftrightarrow{p_i})) + P_{del}$$
$$S'_i = S_{i-1} - \log(\overrightarrow{p}(\text{␣}T_i | R_{i-1}) \cdot \overleftrightarrow{p_i}) + P_{ins}$$

As stated earlier, the penalties are only used if they correspond to changes that are not originally in the given text. Figure 1 shows an example illustrating these equations.

Penalty optimization: The penalties P_{ins} and P_{del} are optimized using a development set of mistokenized sequences and their ground truth. We simulate a beam search assuming that the left context is always predicted correctly, and that the procedure takes a decision after processing the next two characters. Given the ground truth string Q, for every non-space character Q_i and its previous non-space character Q_j, the space probability p_s and non-space probability p_n are computed:

$$p_s = \overrightarrow{p}(\text{␣}Q_i | Q_{1:j}) \cdot \overrightarrow{p}(Q_{i+1} | Q_{1:j}\text{␣}Q_i) \cdot \overleftrightarrow{p_{i'}}$$
$$p_n = \overrightarrow{p}(Q_i | Q_{1:j}) \cdot \overrightarrow{p}(Q_{i+1} | Q_{1:j}Q_i) \cdot (1 - \overleftrightarrow{p_{i'}})$$

where i' is the position of the non-space character corresponding to Q_i. These equations are for BID; the last term in both equations is excluded for UNI.


$\overleftrightarrow{p}$
Q
R
$\overleftrightarrow{p_1}$ $\overleftrightarrow{p_2}$ $\overleftrightarrow{p_3}$ $\overleftrightarrow{p_4}$ $\overleftrightarrow{p_5}$ $\overleftrightarrow{p_6}$ $\overleftrightarrow{p_7}$ $\overleftrightarrow{p_8}$ $\overleftrightarrow{p_9}$
h e l ␣ l o w ␣ o r d
R_1 R_2 R_3 R_4 R_5

$$S_6 = S_5 - \log(\overrightarrow{p}(\text{w}|\text{hello}) \cdot (1 - \overleftrightarrow{p_6}))$$
$$S'_6 = S_5 - \log(\overrightarrow{p}(\text{␣w}|\text{hello}) \cdot \overleftrightarrow{p_6}) + P_{\text{ins}}$$
$$S_4 = S_3 - \log(\overrightarrow{p}(\text{l}|\text{hel}) \cdot (1 - \overleftrightarrow{p_4})) + P_{\text{del}}$$
$$S'_4 = S_3 - \log(\overrightarrow{p}(\text{␣l}|\text{hel}) \cdot \overleftrightarrow{p_4})$$

Figure 1: A snapshot of one beam fixing the string with typo "hel low ord", as explored by bidirectional beam search. For the beam R_5 ="*hello*", the equations for S_6, S'_6 show the updated scores for the two new states of not inserting a space and inserting a space, respectively. For the earlier beam R_3 ="*hel*", the equations for S_4, S'_4 show the updated scores for deleting the space and keeping it, respectively.

If the space is present in the input sequence, the scores S and S' of the candidate sequences without and with a space are:

$$S = -\log p_n + P_{del}$$
$$S' = -\log p_s$$

The space gets deleted if $P_{del} < \log p_n - \log p_s$. If the space is not present in the input, then:

$$S = -\log p_n$$
$$S' = -\log p_s + P_{ins}$$

The space gets inserted if $P_{ins} < \log p_s - \log p_n$.

We perform a grid search on P_{ins} and P_{del} in the range $[0, 20]$ with step size 0.1, and take the combination that maximizes the sequence accuracy defined in Section 5.2.1. We optimize sequence accuracy instead of F-score, because that gives better results on benchmarks with very few errors, where the model must be very conservative.

3.3 Baseline approaches

We evaluate the following three baselines. We also tested a greedy algorithm but omit the results because it consistently performed much worse than the dynamic-programming baseline.

Dynamic programming bigram model: This baseline uses the Viterbi algorithm (Viterbi, 1967) with a word bigram model. First, all possible words (substrings of length ≤ 20 with non-zero unigram frequency in the training data) are located in the sequence without spaces. The states of the Viterbi algorithm are equivalent to the words. A transition between two states is possible if the next word starts at the end of the first word. State transition probabilities are determined by a combination of a unigram and a bigram model, with probabilities p_{uni} and p_{bi} estimated on Wikipedia:

$$p(w_{i+1}|w_i) = \frac{1}{2}(p_{uni}(w_{i+1}) + p_{bi}(w_{i+1}|w_i))$$

The output is the most likely segmentation of the sequence into words.

Wordsegment: *Wordsegment*[1] is an open-source library, based on Halpern (2015), that uses unigram and bigram frequencies to segment words.

Google: To compete with a commercial spell checker, we copy the erroneous sentences into a Google document[2] and accept all suggested edits. We also evaluated the widely used tools Hunspell, TextRazor, and Grammarly, as well as the OCR post-correction system Natas (Hämäläinen and Hengchen, 2019). However, for our datasets, Google yielded the best corrections (for both tokenization and spelling errors).

4 Datasets and training

4.1 Datasets

ACL anthology corpus: The ACL corpus is extracted from scientific articles published between 1965 and 2012 (Bird et al., 2008). The publications were scanned and parsed using OCR, and hence have many typical OCR errors. We manually corrected the tokenization and spelling of 500 sequences for development and penalty optimization, and 500 sequences as a test set.

arXiv: We used the benchmark generator from Bast and Korzen (2017) to generate the text from 910,000 articles from arXiv (parsed from LaTeX files and serving as our ground truth) as well as the text extracted from the corresponding PDF files (using *pdftotext* from FooLabs (2014)). The files were split into paragraphs, which were matched with the ground truth by searching for a text span that differed from the paragraph only by spaces. We thus obtain 64,965,651 sequences for training, and 10,000 sequences each for penalty optimization, development and test.

Wikipedia: We extracted the articles from Wikipedia[3] using WikiExtractor (Attardi, 2017), and split them into sentences using NLTK (Bird et al., 2009). We did a pre-processing to remove sentences that were incomplete or contained markup. We thus obtain 43,103,197 sequences for training, and 10,000 sequences each for penalty optimization, development and test.

We verify the quality of the generated ground truths for Wikipedia and arXiv on 100 sequences from the development sets. Both ground truths have few errors: The 100 sequences from Wikipedia contain no tokenization errors and two spelling errors. The 100 sequences from the arXiv ground truth contain three tokenization errors and no spelling errors.

4.2 Error injection

We inject different types of errors into clean text for two purposes: (1) To create noisy training data. (2) To create large synthetic benchmarks with OCR errors and spelling errors (see Section 5.1).

OCR errors: We consider three kinds of OCR errors: tokenization errors, hyphenation errors (a spurious hyphen due to hyphenation at the end of a line), and character replacement errors (insertions, replacements or deletions of one or more characters, where no spaces are involved).

We estimate the probabilities for these errors on our ACL development set. For hyphenation errors, we divide the total number of spurious hyphens by the number of hyphenable tokens. For space insertions, space deletions, and character replacements, we compute the error rates (number of errors divided by the number of characters) per span of l continuous tokens.

We derive the rules for character replacements and their relative frequency by a comparison of the ACL corpus with a cleaned version[4], and from the ICDAR 2017 and ICDAR 2019 OCR post-correction benchmarks (Chiron et al., 2017; Rigaud et al., 2019).

To inject errors, we first hyphenate each token with the estimated probability of 3.5 % using *PyHyphen*.[5] We pick each token as the beginning of an erroneous span with the estimated probability of 5.8 %. We sample the span length l, space insertion, space deletion and character replacement rates from the collected data, and introduce errors

[1]https://pypi.org/project/wordsegment/

[2]https://docs.google.com, access. Sept. 8, 2021

[3]https://dumps.wikimedia.org/enwiki, dump of June 20, 2019

[4]https://web.eecs.umich.edu/~lahiri/acl_arc.html, we found the corrections to be incomplete, and therefore do not use the cleaned corpus for training or evaluation, but to derive statistics about typical OCR errors.

[5]https://pypi.org/project/PyHyphen/

into the next l tokens accordingly. Character replacements are sampled following their frequency in the ACL and ICDAR datasets. No tokenization errors are injected for the training data.

Human errors: We use another model to inject human errors. Each word gets replaced by a misspelling from a typo collection with 10 % probability. The collection contains 228,414 spelling errors from Peter Norvig[6], Twitter[7] and Hagiwara and Mita (2020), which we split equally into a training/development set and a test set. For the training data, we also generate random spelling errors (character insertions, deletions, replacements or swaps). For the synthetic benchmarks, we inject on average p_t tokenization errors per token, with a constant p_t.

4.3 Models training

We use the two beam search approaches described in Section 3.2: a unidirectional model (UNI), and the unidirectional model combined with the bidirectional sequence labeling model described in Section 3.1 (we call this combination BID). For each of these, we train two variants: one using a combination of the clean *arXiv ground truth* (from the LaTeX source files) and *Wikipedia* datasets from Section 4.1, and the other using the same datasets, but with OCR and spelling error noise injected as described in the previous subsection.

For each approach, we optimize the penalties P_{ins} and P_{del} on the penalty optimization set for every benchmark with spaces; see Section 4.1. Additionally, we consider a model with fixed (benchmark-independent) penalities, averaged across the five benchmarks with spaces. When there are no spaces (which is trivial to detect), we set $P_{ins} = P_{del} = 0$. In summary:

UNI	UNI trained on clean text
UNI+	UNI trained on noisy text
BID	BID trained on clean text
BID+	BID trained on noisy text
BID+ The One	BID+ with fixed penalties

The models were trained for one epoch, which took 86 hours for the unidirectional and 144 hours for the bidirectional models on a NVIDIA Titan X GPU. The training was performed using the Adam optimization algorithm (Kingma and Ba, 2015), with learning rate 0.001, and mini-batch size 128. The sequences were cut after 256 characters, while shorter sequences were padded with EOS symbols that got masked in the loss function. The models are implemented using TensorFlow (Abadi et al., 2015). The unidirectional language model has 67.7 % character accuracy, 88.8 % top-5 character accuracy and 1.099 categorical cross-entropy.

5 Evaluation

5.1 Benchmarks

We evaluate our methods on six benchmarks with different kinds of spelling and space errors. The datasets and ground truths behind these benchmarks, as well as the division into test, development, and training set, are described in Section 4.1. We here describe the corrupt sequences used as inputs to our models.

ACL: The 500 test sequences from the ACL anthology dataset are considered. This benchmark contains many tokenization and OCR errors, often in combinations that are very hard to fix.

arXiv OCR: The 10,000 ground truth sequences from the *arXiv* test set are considered. We create corrupt sequences by injecting OCR errors into the ground truth according to our noise model described in Section 4.2 (this includes tokenization errors). We created this benchmark in order to have a larger benchmark than ACL, but with similar properties.

arXiv pdftotext: The same 10,000 ground truth sequences are considered, but corrupt sequences taken from the output of *pdftotext* on the corresponding PDFs. This benchmark has no spelling errors and few tokenization errors, so it is hard to make the necessary few corrections, yet avoid false positives.

Wikipedia (three variants): For the corrupt sequences of *Wiki+* and *Wiki+ no*␣, we inject spelling errors into the Wikipedia ground truth using the human error model described in 4.2. For *Wiki+*, we inject tokenization errors with a realistic (see Section 1.1) rate of $p_t = 0.01$ per token. For *Wiki+ no*␣, we remove all spaces, which is the scenario from previous work and the *word segmentation* problem. *Wiki* has no spelling errors and a tokenization error rate of $p_t = 0.1$, similar as in the PDF extraction scenario.

Statistics of the tokenization errors in the six benchmarks are given in Table 1. We have also

[6]https://norvig.com/ngrams/spell-errors.txt
[7]http://luululu.com/tweet/typo-corpus-r1.txt

Benchmark	Sequences	Erroneous	Spurious	Missing
ACL	500	190	1,160	297
arXiv OCR	10,000	3,570	14,242	3,798
arXiv pdftotext	10,000	1,274	2,355	594
Wiki	10,000	6,502	7,681	7,261
Wiki+	10,000	1,310	726	727
Wiki+ no ␣	10,000	9,590	0	141,750

Table 1: Statistics of the tokenization repair benchmarks. Erroneous = sequences with tokenization errors, Spurious = spurious spaces, Missing = missing spaces.

Benchmark	Sequences	Tokens	TE	SE	ME
ACL	500	12,990	914	451	143
arXiv OCR	1,000	22,446	1,073	571	133
Wiki+	1,000	15,369	215	901	19
Wiki+ no ␣	1,000	15,369	14,192	0	1,133

Table 2: Statistics of the spelling benchmarks. Tokens = number of ground truth tokens, TE = tokenization errors, SE = spelling errors, ME = mixed errors.

evaluated our approaches on the ICDAR benchmarks (Section 4.2), but do not report them due to severe issues with the ground truth.[8]

5.2 Metrics

5.2.1 Tokenization repair

Given a corrupt input text C, a ground truth text T and a predicted text P, a tokenization repair algorithm predicts a set of space insertions and deletions that ideally would transform C into T. We use two metrics for the evaluation: F-score and sequence accuracy.

F-score: We define $\mathrm{edits}(A, B)$ as the space insertions and deletions that transform A into B. If we let $\mathcal{C} = \mathrm{edits}(C, T)$ be the ground truth edit operations and $\mathcal{P} = \mathrm{edits}(C, P)$ the predicted edit operations, the number of true positives is $\mathrm{TP} = |\mathcal{C} \cap \mathcal{P}|$, the number of false positives is $\mathrm{FP} = |\mathcal{P} \setminus \mathcal{C}|$ and the number of false negatives is $\mathrm{FN} = |\mathcal{C} \setminus \mathcal{P}|$. The F-score is computed as:

$$F(T, C, P) = \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Sequence accuracy: The sequence accuracy is the fraction of sequences that are corrected completely ($P = T$).

5.2.2 Spelling correction

We use a word correction F-score as a metric for spelling correction. We compute the longest common token subsequence of C with T (that is, $C \cap T$), and P with T (that is, $P \cap T$), and define erroneous tokens $E = T \setminus C$, error-free tokens $F = C \cap T$, and correctly predicted tokens $S = P \cap T$. Then $\mathrm{TP} = |E \cap S|$, $\mathrm{FP} = |F \setminus S|$ and $\mathrm{FN} = |E \setminus S|$, and the F-score is computed as above.

We classify each token in E as *tokenization error* if its correction involves only space edits, *spelling error* if only non-space edits, and *mixed* otherwise. See Table 2 for the distribution of the three error types in the benchmarks with spelling errors.

5.3 Main results

Table 3 provides F-scores and sequence accuracies for all our methods on all benchmarks. We use a beam size of $b = 5$ for all beam search approaches; increasing this to $b = 10$ has shown only minimal improvements while doubling the running time. The average running time of our tokenization repair, measured on an NVIDIA Titan X GPU, is 2.0 sec/KB for UNI and 2.6 sec/KB for BID.

The main takeaway from Table 3 is that the bidirectional models beat all four baselines by a wide margin on all benchmarks. They are also better than their unidirectional counterparts trained on the same data. We remark that it is not obvious that the bidirectional methods are the best, which might be the reason why previous work used unidirectional methods. A unidirectional model has the advantage that the tokenization errors are incrementally fixed from left to right, so that the language model predictions can be based on text that is (almost) free from such errors. However, the text after the current position has not yet been repaired, so that predictions from the other direction are based on text with tokenization errors. Using these predictions actually *deteriorates* the quality of the unidirectional methods. Our trick was to combine a unidirectional model that makes use of the space information with a bidirectional model that disregards all space information and thus does not have the aforementioned problem.

The other important takeaways from Table 3 are as follows. The results for previous work are discussed in Section 5.4.

• Tokenization repair is harder when there are *spelling* errors. This is especially pronounced for the ACL benchmark, which has many passages so deformed by OCR errors that there is simply not enough information to reconstruct the correct spacing; see Section 5.5. When there are spelling

[8]Many corrections were missing and the word order was often significantly changed. As a relatively minor complication, some of the text uses old language and symbols not encountered in our training data.

	F-score						Sequence accuracy					
	ACL	arXiv OCR	arXiv pdftotext	Wiki	Wiki+	Wiki+ no ␣	ACL	arXiv OCR	arXiv pdftotext	Wiki	Wiki+	Wiki+ no ␣
Do nothing	0.0	0.0	0.0	0.0	0.0	0.0	62.0	64.3	87.3	35.0	86.9	4.1
Dyn. Progr.	57.4	62.2	27.5	92.6	33.0	98.0	40.2	52.9	68.1	86.2	68.6	68.6
Wordsegment	55.2	58.2	18.7	59.1	9.9	91.1	32.2	52.0	58.3	41.1	32.0	32.0
Google	54.7	60.8	2.5	74.0	66.3	12.4	67.8	75.8	86.0	58.4	90.6	9.1
UNI	83.6	90.6	76.5	98.2	85.8	98.3	70.8	84.5	93.2	95.2	96.0	72.5
UNI+	86.0	96.1	73.5	98.3	92.7	99.1	77.6	92.8	93.4	95.7	98.0	85.3
BID	88.7	94.0	**85.1**	**99.0**	86.5	98.9	76.0	87.7	**94.8**	**97.3**	96.2	80.2
BID+	89.6	**97.5**	84.6	98.7	**93.7**	**99.4**	**79.8**	94.1	94.2	96.8	**98.3**	**89.0**
BID+ The One	**90.6**	**97.5**	81.8	98.9	91.5	**99.4**	79.6	**94.2**	94.1	97.2	97.7	**89.0**

Table 3: Micro-averaged F-scores and sequence accuracies in percent for all models (baselines, existing tools, unidirectional, bidirectional) and the six benchmarks from Section 5.1. The best results for each benchmark are shown in bold. All sequence accuracy differences larger than 0.5 % in the table are statistically strongly significant ($p < 0.01$ with a paired two-sided randomization test). Google's tokenization repair was evaluated on 500 sequences from ACL and 1,000 sequences for all other benchmarks.

errors, training on text with spelling errors (UNI+ and BID+) is crucial, as it enhances the results significantly.

• Tokenization is also harder when there are more *space* errors, yet previous work chose to remove all spaces from the text. Removing all spaces simplifies the approaches (one only has to predict space insertions, no space deletions), but the price is much worse results when there were actually only few tokenization errors; see the sequence accuracies of *Wiki+ no ␣* vs. *Wiki+*.

• The *arXiv pdftotext* benchmark is hard, because $87.3\,\%$ of the sequences have no errors and it is hard to correct the other $12.7\,\%$ without inserting mistakes into the error-free sequences. Indeed, Google is very conservative on this benchmark and suggests almost no corrections (hence the very low F-score). Wordsegment is bolder, with a result that is much worse than doing nothing. UNI and BID improve very significantly over doing nothing.

• The results for "BID+ The One" are close to the results of the best models with penalties tailored to the respective benchmark. This shows that one model works well on different datasets, without having to optimize the penalties for each dataset.

Table 4 shows the capabilities of Google's error correction (the best among all the existing tools, see Section 3.3) with and without our tokenization repair on our four benchmarks with spelling mistakes. We removed internal punctuation in tokens that were merged by our method or the oracle, to help Google to correct more errors. The main takeaway is that Google can correct substantially more errors when our tokenization repair is run as a preprocessing. The reason is that once it is clear which characters form a word, it is much easier to figure out the correct word. Furthermore, the upper bound by the Oracle+Google approach in Table 4 shows that using our method as a pre-processing reaches near optimal results for all benchmarks, except for the ACL benchmark which has severe errors. The difference in corrected spelling and mixed errors between Google and Oracle+Google on the *Wiki+* benchmark is small, because there are only very few mixed errors.

Note that there are *two* reasons why the overall F-score of BID+Google is higher than for Google alone: because our tokenization repair fixes many more tokenization errors than Google and because our tokenization repair helps Google to fix more spelling errors.

Benchmarks	ACL	arXiv OCR	Wiki+	Wiki+ no ␣
Error distribution	61-30-9	60-32-8	19-79-2	93-0-7
Percentage of corrected spelling and mixed errors				
Google	13.5 %	16.5 %	75.0 %	4.0 %
BID+Google	18.4 %	22.2 %	75.2 %	81.2 %
Oracle+Google	19.9 %	22.0 %	75.3 %	82.1 %
F-score for all errors combined				
Google	47.7 %	56.7 %	79.0 %	8.2 %
BID+Google	66.6 %	76.3 %	83.4 %	98.5 %
Oracle+Google	75.4 %	78.2 %	84.1 %	98.9 %

Table 4: Percentages of spelling errors corrected by Google only, Google after our tokenization repair (with BID+), and Google after perfect tokenization repair. The error distribution is given as three percentages: tokenization errors, spelling errors, mixed errors.

Approach	F-score	Seq.acc.
Doval and Gómez-Rodríguez (2019)	99.6 %	92.2 %
UNI	99.6 %	90.7 %
BID	**99.8** %	**94.2** %
Nastase and Hitschler (2018)	40.3 %	19.0 %
BID+	**89.6** %	**79.6** %

Table 5: Evaluation on the English benchmark by Doval and Gómez-Rodríguez (2019) and the ACL benchmark.

5.4 Evaluation of previous work

The results for the best previous work are shown in Table 5. These works did not provide sufficient material to run their methods on our benchmarks. Instead, we ran our approaches on their benchmarks.

The approach from Doval and Gómez-Rodríguez (2019) is an instance of UNI. The BID model performs better than their model and our UNI, which shows the improvement by the bidirectional component. Figures are high for all three models because their benchmark is unrealistic: it has no spelling errors and all spaces are removed. Our results in Table 3 show that the problem is much harder with spelling errors (in particular, see the low sequence accuracy for UNI in column *Wiki+ no* ␣), and that results are much worse when not making use of existing spaces (compare the F-score and sequence accuracy for columns *Wiki+* and *Wiki+ no* ␣).

The results from Nastase and Hitschler (2018) on the ACL corpus are very weak in comparison.[9]

5.5 Error analysis

Our interactive web application under `https://ad.cs.uni-freiburg.de/publications` allows a detailed error analysis for all benchmarks and methods. We here list our most important findings.

Methods trained without spelling errors have a strong tendency to split words with a typo because there is no meaningful continuation (e.g., "*unwnted pregnancies*" is wrongly repaired to "*unw nted pregnancies*") and to wrongly merge misspelled words when they happen to form a correct word (e.g., if "*as well*" is mistyped as "*s well*", it is wrongly repaired to "*swell*").

The ACL benchmark has many passages that are hard to correct given only the input text. In particular: wrong reading order due to imperfect text extraction from the PDFs (e.g., "*Prepa- Seasoning ration*"), ambiguous formulas (e.g., does the text "*nik*" come from "$n_i\, k$" or "n_{ik}"), and very deformed words (e.g., "*(.hme:e "s*" with ground truth "*Chinese*"). The *arXiv OCR* benchmark has, by construction, similar problems, except the wrong reading order; hence the better figures in Table 3.

For the *arXiv pdftotext* benchmark, many of the unresolved errors are in formulas (often involving unusual mathematical symbols) and unconventionally written words (e.g. "*bench mark*"). Many of these could be considered as ground truth errors.

On Wikipedia, our method struggles mostly with compound words (e.g., "*offseason*"), entity names (Baseball team "*Waikiki BeachBoys*" vs. rock band "*The Beach Boys*"), foreign words and inconsistent punctuation. The few remaining errors were due to abbreviations, measuring units or rare symbols.

6 Conclusion

We identified three key ingredients of high-quality tokenization repair, all missing from previous work: bidirectional models, training on text with spelling errors, and making use of the space information already present. Our methods also improve existing spell checkers, by helping them to identify which characters form a word.

There is still room for improvement, especially in a scenario where many spelling (or OCR) errors and tokenization errors come together. However, we show that the remaining errors are hard, with many ambiguous situations. We paid attention to practical running times and all our methods have linear complexity. A carefully trained Transformer model (Vaswani et al., 2017), however, has the potential to achieve comparable results with much faster running times (Walter, 2021).

It remains an open problem whether tokenization repair and spelling correction can be solved "together" both efficiently and with high quality. Our own methods can be extended to also correct spelling errors, but only with very large beam sizes and impractically large running times. Existing models that correct both types of errors (and sometimes even grammar errors) simultaneously fare poorly in comparison to the results we presented.

Acknowledgement

We thank Claudius Korzen for providing the arXiv dataset, and Markus Näther for helpful discussions.

[9] Nastase and Hitschler (2018) report an F-score of 95 %, whereas Table 5 states 40.3 %. There are three reasons for this: (1) they remove all spaces and evaluate how many words they can restore, including words that had no error initially; (2) they evaluate on newer documents which have less OCR errors; (3) they make many errors around punctuation, but their word-level evaluation does not punish this.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org.

Yvonne Adesam, Dana Dannélls, and Nina Tahmasebi. 2019. Exploring the Quality of the Digital Historical Newspaper Archive KubHist. In *Proceedings of the Digital Humanities in the Nordic Countries 4th Conference, Copenhagen, Denmark*, CEUR Workshop Proceedings.

Abdulrahman Almuhareb, Waleed Alsanie, and Abdulmohsen Al-Thubaity. 2019. Arabic word segmentation with long short-term memory neural networks and word embedding. *IEEE Access*, 7:12879–12887.

Giuseppe Attardi. 2017. WikiExtractor: A tool for extracting plain text from Wikipedia dumps. `https://github.com/attardi/wikiextractor`.

Hannah Bast and Claudius Korzen. 2017. A Benchmark and Evaluation for Text Extraction from PDF. In *ACM/IEEE Joint Conference on Digital Libraries, JCDL, Toronto, ON, Canada*, pages 1–10. IEEE Computer Society.

Steven Bird, Robert Dale, Bonnie J. Dorr, Bryan R. Gibson, Mark Thomas Joseph, Min-Yen Kan, Dongwon Lee, Brett Powley, Dragomir R. Radev, and Yee Fan Tan. 2008. The ACL anthology reference corpus: A reference dataset for bibliographic research in computational linguistics. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*. European Language Resources Association.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly.

Guillaume Chiron, Antoine Doucet, Mickaël Coustaty, and Jean-Philippe Moreux. 2017. ICDAR2017 competition on post-ocr text correction. In *14th IAPR International Conference on Document Analysis and Recognition, ICDAR 2017, Kyoto, Japan, November 9-15, 2017*, pages 1423–1428. IEEE.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Rui Dong and David Smith. 2018. Multi-input attention for unsupervised OCR correction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2363–2372. Association for Computational Linguistics.

Yerai Doval and Carlos Gómez-Rodríguez. 2019. Comparing neural- and N-gram-based language models for word segmentation. *Journal of the Association for Information Science and Technology*, 70:187–197.

FooLabs. 2014. Xpdf: A PDF Viewer for X.

Alex Graves. 2013. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850.

Masato Hagiwara and Masato Mita. 2020. GitHub typo corpus: A large-scale multilingual dataset of misspellings and grammatical errors. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6761–6768, Marseille, France. European Language Resources Association.

Orit Halpern. 2015. *Beautiful data: A history of vision and reason since 1945*. Duke University Press.

Mika Hämäläinen and Simon Hengchen. 2019. From the Paft to the Fiiture: a Fully Automatic NMT and Word Embeddings Method for OCR Post-Correction. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing, RANLP 2019, Varna, Bulgaria*. INCOMA Ltd.

Weipeng Huang, Xingyi Cheng, Kunlong Chen, Taifeng Wang, and Wei Chu. 2020. Towards fast and accurate neural chinese word segmentation with multi-criteria learning. In *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 2062–2072. International Committee on Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR , San Diego, CA, USA, Conference Track Proceedings*.

Ido Kissos and Nachum Dershowitz. 2016. OCR error correction using character correction and feature-based word classification. In *12th IAPR Workshop on Document Analysis Systems, DAS 2016, Santorini, Greece, April 11-14, 2016*, pages 198–203. IEEE Computer Society.

Karen Kukich. 1992. Spelling Correction for Telecommunications Network for the Deaf. *Communications of the ACM*, 35:80–90.

Ji Ma, Kuzman Ganchev, and David Weiss. 2018. State-of-the-art chinese word segmentation with bi-LSTMs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4902–4908. Association for Computational Linguistics.

Mark F. Medress, Franklin S. Cooper, James W. Forgie, C. C. Green, Dennis H. Klatt, Michael H. O'Malley, Edward P. Neuburg, Allen Newell, Raj Reddy, H. Barry Ritea, J. E. Shoup-Hummel, Donald E. Walker, and William A. Woods. 1977. Speech understanding systems. *Artif. Intell.*, 9(3):307–316.

Mladen Mikša, Jan Šnajder, and Bojana Dalbelo Bašic. 2010. Correcting Word Merge Errors in Croatian Texts. *The Seventh International Conference on Formal Approaches to South Slavic and Balkan Languages*.

Vivi Nastase and Julian Hitschler. 2018. Correction of OCR Word Segmentation Errors in Articles from the ACL Collection through Neural Machine Translation Methods. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC, Miyazaki, Japan*. European Language Resources Association (ELRA).

Thi-Tuyet-Hai Nguyen, Adam Jatowt, Mickaël Coustaty, Nhu-Van Nguyen, and Antoine Doucet. 2019. Post-OCR error detection by generating plausible candidates. In *2019 International Conference on Document Analysis and Recognition, ICDAR 2019, Sydney, Australia, September 20-25, 2019*, pages 876–881. IEEE.

Thi-Tuyet-Hai Nguyen, Adam Jatowt, Nhu-Van Nguyen, Mickaël Coustaty, and Antoine Doucet. 2020. Neural machine translation with BERT for post-OCR error detection and correction. In *JCDL '20: Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020, Virtual Event, China, August 1-5, 2020*, pages 333–336. ACM.

Kai Niklas. 2010. Unsupervised post-correction of OCR errors. *Diploma thesis. Leibniz Universität Hannover*.

Christophe Rigaud, Antoine Doucet, Mickaël Coustaty, and Jean-Philippe Moreux. 2019. ICDAR 2019 competition on post-OCR text correction. In *2019 International Conference on Document Analysis and Recognition, ICDAR 2019, Sydney, Australia, September 20-25, 2019*, pages 1588–1593. IEEE.

Sandeep Soni, Lauren F. Klein, and Jacob Eisenstein. 2019. Correcting Whitespace Errors in Digitized Historical Texts. In *Proceedings of the 3rd Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature, LaTeCH@NAACL-HLT, Minneapolis, MN, USA*, pages 98–103. Association for Computational Linguistics.

Kazem Taghva and Eric Stofsky. 2001. OCRSpell: an interactive spelling correction system for OCR errors in text. *Int. J. Document Anal. Recognit.*, 3(3):125–137.

Xiang Tong and David A. Evans. 1996. A statistical approach to automatic OCR error correction in context. In *Fourth Workshop on Very Large Corpora, VLC@COLING 1996, Copenhagen, Denmark, August 4, 1996*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269.

Sebastian Walter. 2021. Tokenization repair using transformers. `https://ad-blog.cs.uni-freiburg.de/post/tokenization-repair-using-transformers`. Accessed: 2021-09-07.