Nils Diewald/Marc Kupietz/Harald Lüngen

TOKENIZING ON SCALE

Preprocessing large text corpora on the lexical and sentence level

Abstract When comparing different tools in the field of natural language processing (NLP), the quality of their results usually has first priority. This is also true for tokenization. In the context of large and diverse corpora for linguistic research purposes, however, other criteria also play a role – not least sufficient speed to process the data in an acceptable amount of time. In this paper we evaluate several state-of-the-art tokenization tools for German – including our own – with regard to theses criteria. We conclude that while not all tools are applicable in this setting, no compromises regarding quality need to be made.

Keywords Corpora; tokenization; German; software

1. Introduction

Tokenization, that is, the segmentation of texts into lexical units, is a fundamental preprocessing step for lexicographic work with corpus linguistic resources. Although tokenization is one of the simpler tasks in these processing steps, it is often critical because early errors can affect all analyses and procedures based on it. Accordingly, high accuracy in tokenization is usually the outstanding criterion in the evaluation of tools used for this purpose. Depending on the area of application, other criteria may also play an important role in the evaluation of tokenization tools. The evaluation presented here is based on a scenario in research data preparation, more precisely: the tokenization of DeReKo, the German Reference Corpus. With currently more than 50 billion running words, it is a very large and constantly growing linguistic data resource, for which not only high accuracy is relevant for tokenization, but also a speed that allows the resource to be processed in an acceptable time. Other criteria in this scenario is the extensibility of the language model for new linguistic phenomena and the adaptability for new or different corpora. Also important is the permanent maintainability of the tools and the reproducibility of its results for research. In this article, we evaluate different tools for tokenizing German text data with a special focus on the scenario outlined here. Furthermore, we include sentence segmentation in our consideration, since it is often performed in the same processing step. We include two of our own implementations in the evaluation and compare them with several off-the-shelf tools that we consider state-of-the-art.

2. Tokenization

In corpus technology, tokens represent basic lexical units which are indexed and can be addressed in search queries to the corpus. In fact it is sometimes difficult to query for characters that are delimiters in tokenization. The unsegmented characters of a corpus text are considered its primary data, and its tokenization serves as a basic layer for searching and for higher-level analyses provided as annotations, such as part-of-speech tagging, named entity recognition, syntactic parsing, or anaphora resolution. The tokenization scheme is therefore crucial for the analyses that can be expressed and represented on the higher levels. Furthermore, tokens are the basis for the calculation of the corpus size and statistical measures based on the corpus size and/or token frequencies.

For languages with alphabetical writing systems that use spaces to mark word boundaries (such as German), a simple tokenization algorithm consists of using these spaces and punctuation symbols of a text as delimiters and to consider the resulting strings between them and the punctuation symbols themselves as tokens. This method leads to meaningful tokens in the majority of cases already (s. sec. 4.3).

The bulk of the difficulties with tokenization arises from the potential ambiguity of the space and punctuation characters, as in certain cases they should not be considered token delimiters but instead parts of tokens. Certain multi-word expressions, such as "ad hoc", "bus driver" or "heart attack" in English should under a syntactic perspective be analysed as single lexical tokens that just happen to contain the space character. As for punctuation characters, there are several cases of when the dot '.' does not represent the full stop terminating a sentence e.g. in abbreviations ("etc."; "Fa.", short for *Firma*, 'company'; "bzw.", short for *beziehungsweise*, 'respectively'), in ordinary numbers and enumerations ("1.", "a.)", "B."), as part of email addresses or URLs, and in several other cases (cf. Proisl/Uhrig 2016). Similarly, other punctuation symbols are ambiguous between delimiters and other uses, for instance the hyphen in "Hamburg-München" (naming a distance, leading to three separate tokens) vs. in "Reich-Ranicki" (a surname, i.e. one token), or in the brand name "Yahoo!", the "!" should not be separated. Non-alphabetic characters other than punctuation can also be ambiguous between representing a delimiter vs. a regular character that is part of a token, e.g. in an arithmetic expression like "5+3", the "+" is a delimiter, but in "C++", it should not be separated, also cf. the asterisks in an action word like "*grins*" (from *grinsen*, 'to grin'; separate tokens according to the Tokenization Guidelines of the EmpiriST Shared Task; Beißwenger et al. 2015) vs. in a form like "Lehrer*innen" (gender neutral form for 'teachers'; one token). In turn, there are also cases when strings without delimiters actually contain separate tokens, mostly when spaces are intentionally or unintentionally omitted as in "erhat" for *er hat* ('he has').

In our recent corpora, we have identified five types of phenomena, mostly connected with the internet as a media for the distribution of texts, which pose relatively new challenges for tokenization by introducing additional ambiguities, or significantly increasing the frequency of occurrence of certain known ambiguities.

1) The proliferation of unedited text, i.e. texts that contain sloppy or creative uses of spelling, which can also affect the tokenization, as in contracted forms based on the spoken language e.g. "haste" (→ *hast du*; 'do you have'), "hastes" (→ *hast du es*, 'do/here you have it'), the omission of spaces ("Hänselundgretel", *Hänsel und Gretel*, 'Hansel and Gretel'), the insertion of extra spaces ("Auto Bahn", *Autobahn*, german highway system), or an iterative use of punctuation symbols ("Drogen!!!!", 'Drugs!!!!') (cf. Bartz/Beißwenger/Storrer 2013).
2) The proliferation of computer-mediated communication (CMC) idioms used in social media which besides displaying features of the spoken language as mentioned above, contain new specific uses of punctuational and non-alphabetical characters as in emoticons such as ":-)", addressing terms ("@heiner"), or action words and phrases ("*lach*", from *lachen*, 'to laugh'; "*auf die Nägel blas*", 'to blow on the nails') (cf. Bartz/Beißwenger/Storrer 2013).

3) New, token-internal uses of delimiters in gender-conformant spellings as in German forms like "Lehrer(-in)", "Lehrer:in", "Lehrer/-innen", "Lehrer/innen", "Lehrer_innen", "Lehrer*innen" (gender neutral forms for 'teachers'), "diese*r" (gender neutral form for 'this/these/those'), "Frau*" (gender neutral form for 'woman').
4) Hypertextual features such as hashtags, mentions, filenames, email addresses, URLs, but also XML or HTML markup, Markdown, or WikiCreole source code (see Jurish/Würzner 2013).
5) The huge quantities of text that the internet offers pose severe processing challenges for tokenization in terms of time and space.

Note that different NLP applications or a different focus of linguistic description may require different tokenization strategies. For instance, when the focus is on syntax/parsing, compounds are analysed as one token, whereas for semantic relation or information extraction, it might be relevant to even tokenize the parts of compounds that are spelt as one word (such as "Abgasrückführung", 'exhaust gas recycling'). In fact, different NLP tools often disagree in their tokenization (s. sec. 4.3), and their tokenization strategies cannot independently be considered as correct or incorrect. The interpretation of tokens we adopt is closer to a lexicographic reading and might deviate from the preprocessing in some machine-learning (ML) workflows that are based on dictionary queries and tokenize other units to handle out-of-vocabulary situations. We also do not consider tokenizations into sub-lexical units such as morphemes.

The task of sentence segmentation is closely related to tokenization due to the central role of punctuation symbols and their ambiguity between being a part of a token or terminating a sentence. In fact, tokenization and sentence segmentation are often applied in the same processing step. Besides the ambiguity of punctuation, sentences or sentence-like units might not be delimited by a full stop at all, as for example regularly in the case of headings, but also in instances of sloppy writing in CMC. Sentences represent the basic scope of later syntactic analyses, and a faulty sentence segmentation renders automatic syntactic parsing largely invalid. In corpus technology, the sentence is also the default domain of a query, i. e. when querying for and analysing expressions with multiple parts (as in "ADJ N"), it is crucial that no sentence boundary lies between them. Moreover, various licence-related restrictions refer to the unit *sentence*. Consequently, this unit is of great, not only linguistic, importance in corpus technology.

3. Large Scale Scenario

Our main use case for corpus tokenization represents the preprocessing of the German reference corpus DeReKo (Kupietz et al. 2018). DeReKo has been compiled at the Leibniz Institute for the German Language since 1964 and currently comprises more than 50 billion running words with an annual increase of more than 2 billion. It is used for a broad range of linguistic research on written contemporary German and for this purpose is completely tokenized, and morphologically and syntactically annotated multiple times. For researchers, access is limited (primarily for licensing reasons) to search engines (COSMAS II and KorAP) and further analysis tools. In addition, various derived analysis data such as frequency-based wordlists are offered. These different forms of usage also determine the pragmatic token definition on which DeReKo is based, which tries to be both "linguistically significant and methodologically useful" (Webster/Kit 1992, p. 1106), although compromises must be made for both aspects. This is particularly important to consider with respect to "word" tokens, which must follow a lexicographic definition. Idioms and fixed expressions, which consist of several, possibly even discontinuous units, can be useful in a lexicological sense, but detrimental for search engine use, which is why DeReKo does not take them into account. Consequently, we assume, on the one hand, tokens to be the "minimal unit of investigation" (Chiarcos/Ritz/Stede 2009, p. 35), and, on the other hand, units that are meaningful for representations in syntactic contexts.

With respect to the data, both in terms of size and diversity, there are further pragmatic requirements for tokenization. In our evaluation, we emphasize high processing **speed** with large data volumes and limited resources. In the case of DeReKo, this is important with respect to the constant acquisition of data, but is especially necessary when a complete re-tokenization of the entire corpus is required, for example, to correct systematic errors while maintaining model consistency. The limited technical resources also mean, in our scenario, that the procedures should run on commodity hardware – accordingly, we do not consider possible performance increases through special hardware (such as GPU support) in this analysis.

Furthermore, high **quality** is of obvious importance, since tokenization is the basis of further linguistic analysis steps, and, as Moreau/Vogel (2018, p. 1120) correctly note: "Tokenization errors can be costly performance-wise, as these errors may propagate through the whole processing chain."

A controllable **extensibility** of tokenization is of great importance especially with respect to new phenomena and new, special corpora. In some cases, prior linguistic knowledge can be used for extension, but also specially prepared training corpora. What plays only a minor role for our application scenario, but is primary in many other scenarios, is the **adaptability** to other languages and corpora. Even though we have adapted our tools for other languages, the focus is on German language data.

With regard to the use of processed data for research purposes, other important aspects are the **maintainability** of the solutions and the **reproducibility** of the results. For maintainability, it is necessary that the solutions are available as open source, which is why we restrict the evaluation exclusively to this. Applicability to commodity hardware has already been mentioned but is also central in terms of maintainability. In terms of reproducibility, the results, but also the errors, should be consistent and comparable across genre and domain boundaries. Along with this, a desirable advantage is the reduction of the tokenization step to as few tools as possible. Different tokenizers for heterogeneous data, adapted to different corpus types, genres, or domains, would not only complicate the traceability and thus reproducibility of any tokenization errors, they would also pose a major challenge in the handling and maintainability of the full preparation pipeline.

Not to be considered in our scenario are preprocessing steps that may be necessary with respect to the specific origin of the data. Since the source data for DeReKo usually come in XML or other preprint format, there is no need to perform print-specific preprocessing steps from text typesetting, such as merging hyphenated words at line endings or removing marks for highlighting (cf. Grefenstette/Tapanainen 1994). The same is true for mis-segmentation originating from OCR processes. We also do not take ease of use into account.

In summary, DeReKo tokenization requires fast, accurate and consistent processing of heterogeneous data for different linguistic application purposes on commodity hardware. The tools to consider should be easily extensible and maintainable. Both the scenario outlined here and the evaluation of state-of-the-art approaches represent only a snapshot. At the same time it is an update and an extension of similar studies on tools for different NLP tasks in German (Ortmann/Roussel/Dipper 2019), but with a focus on tokenization. In the future, evaluations will be carried out with regard to changing scenarios and new developments in corpus technology.

4. Evaluation

Comparing different software often brings up issues, especially in a task area that is not clearly defined, as tokenization is. As mentioned at the beginning, the scenarios for which tokenization is used differ considerably, which has an influence on the design and applicability of the software. In fact, comparing off-the-shelf solutions with tailor-made tools is always unfair. Therefore, it should be clearly stated that the following comparisons refer to the presented scenario only. In addition, secondary functions of the different tools are not taken into account, like the return of token classes or normalization, even though they can be essential in other scenarios and can have a negative impact especially with regard to speed. The tools are only tested via command line interfaces, so we do not take into account different programming languages. If no native command line tool exists, we have written minimal wrappers following instructions, which should be taken into account regarding speed comparisons as well. We also consider the tools only with respect to a single language (German), while many tools have a primary focus on cross-language applicability. Furthermore, the hardware and software architecture used has a strong influence on the results. As mentioned in Section 3, our tests disfavour applications that can use dedicated GPU support, for example. We provide the full test suite in the form of a Dockerfile[1] to make it replicable for other users and on other systems. Table 2 (at the end of this article) provides an overview of our evaluation for all tools in the categories presented.

4.1 Tools

The evaluated tools are all available under different open licenses and represent in our eyes the state of the art with respect to tokenization and sentence segmentation for German, although we cannot claim to be exhaustive.

Our tools for token and sentence boundary detection

- **KorAP-Tokenizer**[2] is rule-based and compiles, using the lexical analysis generator framework JFlex,[3] a list of regular expressions into a deterministic finite state automaton that can introduce segment boundaries at terminal nodes. The ruleset is based on Apache Lucene's tokenizer and has been extensively modified. Rulesets are available for English, French and German. KorAP-Tokenizer is used productively for tokenization and (among other tools) for sentence segmentation of DeReKo.
- **Datok**[4] (Diewald 2022) is rule-based and generates an extended deterministic finite state automaton based on a reduced finite state transducer generated by XFST (Beesley/Kart-

1 https://github.com/KorAP/Tokenizer-Evaluation.

2 https://github.com/KorAP/KorAP-Tokenizer.

3 https://jflex.de/.

4 https://github.com/KorAP/Datok.

tunen 2003). The ruleset of KorAP-Tokenizer was translated to XFST for this purpose. The generation is done with Foma (Hulden 2009). Rulesets are only available for German at this time. Datok is currently being evaluated experimentally.

Tools for token and sentence boundary detection

- **BlingFire**[5] is rule-based and compiles a deterministic finite state automaton based on regular expressions, which segments at terminal nodes. The tested model is implemented cross-language with a focus on English.
- **Cutter** (Graën/Bertamini/Volk 2018) is rule-based and recursively applies language-specific and language-independent rules to a text to segment it. Compared to other rule-based tools, Cutter uses a context-free rather than a regular grammar.
- **JTok**[6] is based on cascading regular expressions that segment tokens until they can be assigned to a token class. Rules exist for English, German and Italian.
- **OpenNLP**[7] is a framework that offers tokenizers and sentence segmenters in different models, both based on maximum entropy. In addition, OpenNLP offers SimpleTokenizer, a tool based on simple character class decisions.
- **SoMaJo** (Proisl/Uhrig 2016) is rule-based and applies a list of regular expressions to segment a text. SoMaJo won first place in the competition of the aforementioned EmpiriST 2015 Shared Task for tokenizing German language Web and CMC corpora and has been regularly improved since then. SoMaJo is available specifically for German.
- **SpaCy**[8] is a framework in which the tokenization stage is rule-based and runs in several phases in which the tokens are split into increasingly finer segments. Rulesets are provided for numerous languages. Different models are offered for sentence segmentation: *Sentencizer* is rule-based, *Dependency* performs a syntactic analysis, *Statistical* segments based on a simple statistical model.
- **Stanford Tokenizer**[9] is rule-based, and relies on JFlex (see KorAP-Tokenizer) to compile a deterministic finite state automaton based on a list of regular expressions that can introduce segment boundaries at terminal nodes.
- **Syntok**[10] is rule-based and applies successive separation rules, primarily in the form of regular expressions, to an input string for segmentation. There is both a tokenizer and a sentence segmenter based on it. Syntok was the fastest tokenizer in Ortmann/Roussel/Dipper (2019). Rules exist for Spanish, English, and German.
- **Waste** (Jurish/Würzner 2013) is based on a hidden Markov model in which a pre-segmented stream of (pseudo)tokens are re-evaluated at the boundaries found and classified as to whether they are word-initial or sentence-initial.

[5] https://github.com/microsoft/BlingFire.

[6] https://github.com/DFKI-MLT/JTok.

[7] https://opennlp.apache.org/.

[8] https://spacy.io/.

[9] https://nlp.stanford.edu/software/tokenizer.shtml.

[10] https://github.com/fnl/syntok.

Tools for token boundary detection only

- **Elephant**[11] (Evang et al. 2013) is an ML system for segmentation based on Conditional Random Fields and Recurrent Neural Networks. We evaluate here a wrapper implementation[12] (Moreau/Vogel 2018) that considers only token segmentation and not sentence segmentation, although Elephant provides both.
- **TreeTagger** (Schmid 1994) is a part-of-speech tagger that carries a separate rule-based tokenization tool that also uses a set of regular expressions to segment a text. The tokenizer does not itself introduce markers for sentence boundaries.

Tools for sentence boundary detection only

- **Deep-EOS** (Schweter/Ahmed 2019) is based on different implementations of neural networks with long short-term memory (LSTM), bidirectional LSTM, and convolutional neural networks. It is not based on pre-tokenization and operates directly on character streams.
- **NNSplit**[13] is an ML approach based on a byte-level LSTM neural network.

In the list of tools we compare here, it is striking that rule-based procedures still dominate tokenization even in modern frameworks, although this is decreasing in other areas of NLP. For sentence boundary recognition, on the other hand, ML techniques seem to be slowly replacing rule-based procedures in this area. ML methods have experienced an increase in importance in recent years due to the availability of large corpora and more powerful computers. But deterministic methods have also benefited (albeit to a lesser extent), through the efficient application of arbitrarily large rulesets and almost arbitrarily large lexicons.

4.2 Performance: speed

Tokenization is not only an NLP problem that can be considered relatively simple, but also one that takes little time to process (compared to, e. g., syntactic parsing). Therefore, when evaluating new tokenizers for research, it is uncommon to specify the runtime. This is slowly changing in the context of machine learning, where tokenizers are used as a pre-processing step for training with very large data sets and speed is therefore of greater importance. But when processing very large corpora in a research context, runtime must be taken into account as well.

For the benchmarking, the novel “Effi Briest” by Theodor Fontane in the Project Gutenberg version was used (with a total of 98,207 tokens[14]). The measures correspond to the average value of 100 runs. Since the length of a text can have an impact on performance, a tenfold concatenation of the text was also tested.

Figure 1 compares the speed of all tools we measured in terms of “tokens per millisecond”.[15] Detailed values are listed in Table 1.

[11] https://gmb.let.rug.nl/elephant/.

[12] https://github.com/erwanm/elephant-wrapper.

[13] https://bminixhofer.github.io/nnsplit/.

[14] Based on the information provided by the Unix tool ‘wc -w’.

[15] The test system is an Intel Xeon CPU E5-2630 v2 @ 2.60GHz with 12 cores and 64 GB of RAM.

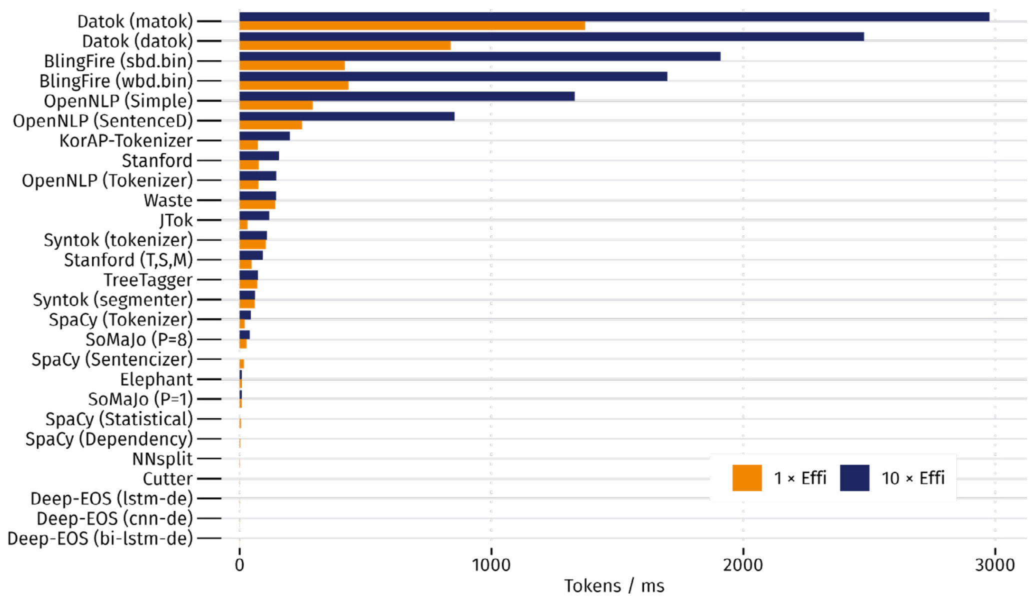

Fig. 1: Speed comparison of all tested tools with different models and configurations

With respect to DeReKo, these times can be extrapolated for individual cores. Our experimental tool Datok (Model “matok”) is the fastest tool in the comparison and could completely tokenize and sentence segment DeReKo within ~13.5h,[16] closely followed by BlingFire with ~33h. KorAP-Tokenizer currently takes ~193h for the same task, just over 8 days. Most other rule-based off-the-shelf tools for tokenization are also in the same range. SoMaJo, which is to be emphasized with respect to German CMC data (s. sec. 4.3), would require about 72 days (without direct parallelization).[17] Cutter was not able to fully segment the large batch size and would have taken just over 4 years to segment DeReKo at small batch sizes. In particular, Deep-EOS and NNsplit show that their use without dedicated hardware (GPU) is not an option for sentence segmentation in our scenario: full processing of DeReKo would take between 1.7 and almost 6.4 CPU years on the hardware used, with this being in addition to tokenization. NNSplit admits to poor performance in terms of CPU-only usage and claims to be twice as fast as SpaCy’s Sentencizer in GPU usage[18]. A similar speed increase should be expected with Deep-EOS.

It can be seen that the speed of tokenization and sentence segmentation is a variable to be considered for large data sets, and not all tools are capable of complete data resegmentation in an acceptable amount of time. It should be noted though that even slower tokenizers like Elephant and SoMaJo (on one core) can still process over 8,000 tokens in one second – and are perfectly adequate for most scenarios and corpus sizes.

4.3 Performance: quality

Even though processing speed is the focus of our review, a high quality of tokenization is of primary importance in the aforementioned scenario. The evaluation of tool quality in an NLP context is typically operationalized via a comparison to a Gold Standard. Already in the

[16] All extrapolations are based on the measured values of “1 x Effi” in Table 1.

[17] SoMaJo supports the use of multiple processor cores, which we have included in our benchmarks, but is less of an issue, as parallelization happens at a higher level in our scenario.

[18] https://bminixhofer.github.io/nnsplit/#benchmark

seemingly simple case of tokenization, however, the definition of what a token is and where its boundaries are leaves room for interpretation (s. sec. 2). Therefore, it is not appropriate to speak of “correct” or “incorrect” for a tokenization – it often depends on the field of application what makes a token an independent entity (s. sec. 3). This has to be taken into account for evaluation related to existing corpora that may follow different guidelines. The same holds for sentence boundaries.

We use the EmpiriST Web and CMC corpora as well as version 2.9 of the German Universal Dependency GSD Corpus (McDonald et al. 2013) in our evaluation of tokenization. We also use the latter for evaluating sentence boundary detection. We rely on the EmpiriST software[19] as the tool for computing F_1 values.

Token boundary detection

All tools except the OpenNLP Simple Tokenizer achieve an F_1 score well above 99% for the UD corpus (s. Tab. 1). For the web corpus, SoMaJo, Cutter, TreeTagger, KorAP-Tokenizer and Datok also achieve more than 99%. With respect to the CMC corpus, SoMaJo, Stanford Tokenizer, TreeTagger, Cutter, KorAP-Tokenizer, and Datok achieve an F_1 score above 95%. Regarding our preference to use only one tool for different corpora, we consider them more suitable for our purposes. Nonetheless, the evaluation of quality is not very meaningful with respect to KorAP-Tokenizer and Datok, since rule-based approaches can be optimized for the evaluation data, and thus in many cases perfect accuracy can be achieved (with limitations, s. sec. 4.6). Accordingly, they cannot be compared with the evaluation within EmpiriST, which was about testing against unknown data.

Tool	V.	Model	UD-GSD (Tokens)	EmpiriST-CMC	EmpiriST-Web	UD-GSD (Sentences)	1 x Effi	10 x Effi
			F_1	F_1	F_1	F_1	T/ms	T/ms
KorAP-Tokenizer	2.2.2		99.45	99.06	99.27	96.87	72.90	199.28
Datok	0.1.5	datok	99.45	98.79	99.21	97.60	614.72	2304.13
		matok					1041.63	2798.78
BlingFire	0.1.8	wbd.bin	99.25	55.85	95.80	-	431.92	1697.73
		sbd.bin	-	-	-	95.90	417.10	1908.87
Cutter	2.5		99.47	96.24	99.38	97.31	0.38	-
JTok	2.1.19		99.56	58.44	98.09	97.92	31.19	117.22
OpenNLP	1.9.4	Simple	95.70	55.26	91.69	-	290.71	1330.23
		Tokenizer (de-ud-gsd)	99.67	65.22	97.58	-	74.65	145.08
		SentenceDet. (de-ud-gsd)	-	-	-	98.51	247.84	853.01
SoMaJo	2.2.0	p=1	99.46	99.21	99.87	97.05	8.15	8.41
		p=8					27.32	39.91

[19] The comparison tool was developed by Stephanie Evert and published under GPL v3. For the evaluation of the sentence boundaries, the segmented sentences are taken instead of single tokens.

Tool	V.	Model	UD-GSD (Tokens)	EmpiriST-CMC	EmpiriST-Web	UD-GSD (Sentent-ces)	1 x Effi	10 x Effi
			F_1	F_1	F_1	F_1	T/ms	T/ms
SpaCy	3.2.3	Tokenizer	99.49	69.94	98.29	-	19.73	44.40
		Sentencizer	-	-	-	96.80	16.94	40.58
		Statistical	-	-	-	97.16	4.90	10.01
		Dependency	-	-	-	96.93	2.24	0.48
Stanford	4.4.0	tokenize	99.93	97.71	98.46	-	75.47	156.24
		tokenize,ssplit, mwt				98.22	46.95	91.56
Syntok	1.4.3	Tokenizer	99.41	70.76	97.50	-	103.90	108.40
		Segmenter	-	-	-	97.50	59.66	61.07
Waste	2.0.20-1		99.55	65.90	98.49	97.46	141.07	144.95
Elephant	0.2.3		99.62	66.96	97.88	-	8.57	8.68
Tree-Tagger	3.2.4		99.52	95.58	99.27	-	69.92	72.98
Deep-EOS	0.1	bi-lstm-de	-	-	-	97.47	0.25	0.24
		cnn-de	-	-	-	97.49	0.27	0.25
		lstm-de	-	-	-	97.47	0.29	0.27
NNSplit	0.5.8		-	-	-	95.55	0.90	0.90

Table 1: Overview of all compared tools and models with their performance measures (best three highlighted in each category)

Moreover, the quality with respect to the applied machine learning tools says less about the implementation or the algorithm than about the corpus used for training. Thus, we compare here "off-the-shelf" solutions with default configurations for the outlined scenario without checking whether a tool trained and adjusted for our purposes would achieve better results.

Taking this expected bias into account, the results show that high tokenization speed does not require any compromises in terms of quality. Furthermore, the comparison of the three evaluated datasets shows that approaches exist that work across genres and thus facilitate the use in the outlined scenario.

Sentence boundary detection

All tools show F_1 values above 95% and can therefore be considered suitable for sentence boundary detection. Stanford and the OpenNLP model show values above 98%. They were also developed and trained using the test corpus. Our tools KorAP-Tokenizer and Datok perform weaker in relation – a readjustment is desirable.

4.4 Extensibility and adaptability

Machine learning-based approaches have the advantage of being easily transferable to other languages or genres in the presence of appropriately annotated corpora. "This is why [Moreau and Vogel] argue that the evaluation of software tools should progressively shift the focus from accuracy in a specific language to robustness and adaptability to a wide range of languages." (Moreau/Vogel 2018, p. 1119). Also, with respect to novel linguistic phenomena (as they occur in various CMC corpora mentioned above), rule-based approaches are in principle inferior, since they cannot deal with unknown phenomena to begin with. However, SoMaJo's win at EmpiriST 2015 shows that a rule-based system designed for extensibility nevertheless offers advantages that can make it competitive in many scenarios. Graën/Bertamini/Volk (2018) even put their main focus regarding their rule-based system Cutter on extensibility and on iterative adaptation to new languages.

The good adaptability of rule-based systems with respect to new language phenomena is largely due to the pattern-like nature of these entities, such as email addresses, emoticons, or XML fragments. These can be well formulated in rule-based terms. Extralinguistic units also occur in other word-segmented languages (Graën/Bertamini/Volk 2018), which makes them *language-independent*. *Language-specific* rules, on the other hand, include general definitions of words and how to deal with punctuation, especially in sentence segmentation. In addition, there are usually lists of common abbreviations (for the correct treatment of periods, e. g. "etc.") and known proper names with non-alphabetic symbols (e. g. "3G+") in that language. As mentioned in Section 4.1, both KorAP-Tokenizer and Datok are based on widely used and well documented frameworks for the rule definition of lexical analysers. Both systems can be adapted with little training, especially with regard to the various lists that can be extended without any knowledge of syntax. By reusing the language-independent rules, the approaches can also be adapted for other languages with manageable effort. This has already been done for KorAP-Tokenizer with respect to English and French.

4.5 Reproducibility and maintainability

A distinct advantage of rule-based systems over machine-learning approaches, especially in the scientific context, is the reproducibility of tokenization. Thus, errors can be traced back to individual rules, which can be easily and likewise systematically corrected, validated with reference to regression tests (cf. Graën/Bertamini/Volk 2018), and versioned. Corrections can be made consistently across the data. Trained machine learning approaches have the disadvantage that an extended corpus must first be created in order to (re-)train the tools. And only very extensive and difficult-to-maintain tests can ensure that re-training does not cause regressions. Rule-based approaches therefore allow for a good balance between correctness and reproducibility in the processing of scientific research data.

In terms of maintainability, data consistency, and runtime, it is also advantageous if token and sentence segmentation can be performed with a single tool, based on the same model.

Also of importance for maintainability are short development cycles. Training and testing machine-learning methods is time-consuming and can often be solved in acceptable time only with special hardware. Compiling complex finite state automata of rule-based approaches can also lead to significant time and resource consumption. Systems like SoMaJo or Cutter can be tested without intermediate steps while providing good extensibility, which simplifies their maintenance. Separating the language model from program code also facilitates maintenance and versioning. In the case of machine-learning, this is usually the case, but rule-based approaches also often have separate language models, for example Cutter, but also (in part) the JFlex-based tools such as Stanford-Tokenizer and KorAP-Tokenizer, and Datok's XFST model.

Tool	Tokens & Sentences	Speed	Quality	Extensibility	Adaptability	Maintainability	Reproducibility
KorAP-Tokenizer	•••	••	•••	•••	••	••	•••
Datok	•••	•••	•••	•••	•	••	•••
BlingFire	••	•••	•	••	••	••	•••
Cutter	•••	-	•••	••••	••••	••••	•••
JTok	•••	••	•	••	•	••	•••
OpenNLP	••	••	•	•	•••	•	-
SoMaJo	•••	•	••••	•••	•	••••	•••
SpaCy	••	•	•	•••	•••	••	••
Stanford	••	••	•••	•••	••	••	•••
Syntok	•••	••	•	•••	••	•	•••
Waste	•••	••	•	•	••	•	-
Elephant	•	•	•	•	••	•	-
TreeTagger	•	••	•••	•••	•	•••	•••
Deep-EOS	•	-	••	•	•••	••	-
NNSplit	•	-	•	•	•••	••	-

Table 2: Overview of all compared tools with ratings in the examined categories

4.6 Limitations

There are some fundamental limitations regarding single-pass finite state systems that need to be considered, nonetheless, and that apply to our approaches.

Graën/Bertamini/Volk (2018) primarily point out long-distance relationships between tokens as a weakness of finite state based systems and as an example refer to the use of the apostrophe in German as a possessive marker for words ending in a phonetic /s/. These apostrophes must belong to the preceding token in the possessive case, but can also represent the end of an expression in simple quotation marks. Without the context of a starting quotation expression, a decision in a finite state-based approach is not possible – accordingly, neither KorAP-Tokenizer nor Datok can make these decisions.

Furthermore, the strict left-longest-match directive means that sometimes valid tokens can never be segmented. An example would be (following general word and URL rules) the string “Go tohttp://google.com/”, in which a space was omitted by mistake and which would currently not be segmented into the expected tokens “Go”, “to” and “http://google.com/” by both KorAP-Tokenizer and Datok, since “tohttp” is considered a valid word token and is a longest-match accordingly. The subsequent tokens would be further segmented as the URL rule would not apply. More limitations concern the processing of emoji sequences, which are difficult to represent in strict finite-state models without character ranges.

5. Summary

Tokenization and sentence boundary detection for texts of word-segmented writing systems belong to the simpler and faster tasks of NLP, and already with naïve approaches good results can be achieved. However, tokenization errors can have a cascading effect on further analysis steps, which is why high quality is of great importance. The emergence of new token types and very large data volumes in the context of unedited, heterogeneous CMC corpora pose further new challenges, especially in research data processing.

In this paper, different tokenizers are compared with respect to this scenario for German in terms of their processing speed, quality, extensibility, adaptability, maintainability and reproducibility. We also present the approaches that are being pursued in building DeReKo.

We believe that processing speed for very large data is a dimension that should not be neglected and that approaches are possible that do not have to compromise on quality with respect to heterogeneous data. In our opinion, the maintainability and reproducibility of process results of rule-based systems represent a further advantage over machine-learning approaches (at least currently), especially in the context of research data processing. But this assessment is only a snapshot: Future developments in this area as well as changes to the scenario described make constant re-evaluations necessary.

We would like to acknowledge the helpful comments of the anonymous reviewers, which clearly contributed to the improvement of the present study.

References

Bartz, T./Beißwenger, M./Storrer, A. (2013): Optimierung des Stuttgart-Tübingen-Tagset für die linguistische Annotation von Korpora zur internetbasierten Kommunikation: Phänomene, Herausforderungen, Erweiterungsvorschläge. In: JLCL 28 (1).

Beesley, K. R./Karttunen, L. (2003): Finite state morphology. (= CSLI Studies in Computational Linguistics). Stanford.

Beißwenger, M./Bartsch, S./Evert, S./Würzner, K.-M. (2015): Richtlinie für die manuelle Tokenisierung von Sprachdaten aus Genres internetbasierter Kommunikation. Guideline document from the Empirikom shared task on automatic linguistic annotation of internet-based communication. In: EmpiriST 2015.

Beißwenger, M./Bartsch, S./Evert, S./Würzner, K.-M. (2016): EmpiriST 2015: A shared task on the automatic linguistic annotation of computer-mediated communication and web corpora. Proceedings of the 10th Web as Corpus Workshop, pp. 44–56.

Chiarcos, C./Ritz, J./Stede, M. (2009): By all these lovely tokens... Merging conflicting tokenizations. Proceedings of the Third Linguistic Annotation Workshop (LAW III), pp. 35–43.

Diewald, N. (2022): Matrix and double-array representations for efficient finite state tokenization. In: Proceedings of the 10th Workshop on Challenges in the Management of Large Corpora (CMLC-10) at LREC 2022. Marseille, pp. 20–26.

Evang, K./Basile, V./Chrupała, G./Bos, J. (2013): Elephant: sequence labeling for word and sentence segmentation. Proceedings of the EMNLP 2013: Conference on Empirical Methods in Natural Language Processing. Seattle.

Graën, J./Bertamini, M./Volk, M. (2018): Cutter – a universal multilingual tokenizer. In: Cieliebak, M./Tuggener, D./Benites, F. (eds.): Swiss text analytics conference, Nr. 2226, pp. 75–81.

Grefenstette, G./Tapanainen, P. (1994): What is a word, What is a sentence? Problems of tokenization. Proceedings of COMPLEX ’94, pp. 79–87.

Hulden, M. (2009): Foma: a finite-state toolkit and library. Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pp. 29–32.

Jurish, B./Würzner, K.-M. (2013): Word and sentence tokenization with Hidden Markov Models. JLCL, 28 (2), pp. 61–83.

Kupietz, M./Lüngen, H./Kamocki, P./Witt, A. (2018): The German Reference Corpus DeReKo: New developments – new opportunities. Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018), pp. 4353–4360.

McDonald, R./Nivre, J./Quirmbach-Brundage, Y./Goldberg, Y./Das, D./Ganchev, K./Hall, K./Petrov, S./Zhang, H./Täckström, O./Bedini, C./Bertomeu Castelló, N./Lee, J. (2013): Universal dependency annotation for multilingual parsing. Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pp. 92–97.

Moreau, E./Vogel, C. (2018): Multilingual word segmentation: training many language-specific tokenizers smoothly thanks to the Universal Dependencies Corpus. Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018). Miyazaki.

Ortmann, K./Roussel, A./Dipper, S. (2019): Evaluating off-the-shelf NLP tools for German. In: Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019). Erlangen, pp. 212–222.

Palmer, D. D./Hearst, M. A. (1997): Adaptive multilingual sentence boundary disambiguation. In: Computational Linguistics, 23 (2), pp. 241–267.

Proisl, T./Uhrig, P. (2016): SoMaJo: state-of-the-art tokenization for German web and social media texts. Proceedings of the 10th Web as Corpus Workshop, pp. 57–62.

Schmid, H. (1994): Probabilistic part-of-speech tagging using decision trees. Proceedings of International Conference on New Methods in Language Processing.

Schweter, S./Ahmed, S. (2019): Deep-EOS: general-purpose neural networks for sentence boundary detection. Proceedings of the 15th Conference on Natural Language Processing (KONVENS). Erlangen.

Webster, J. J./Kit, C. (1992): Tokenization as the initial phase in NLP. Proceedings of the 14th Conference on Computational Linguistics (4), pp. 1106–1110.

Contact information

Nils Diewald
Leibniz-Institut für Deutsche Sprache
diewald@ids-mannheim.de

Marc Kupietz
Leibniz-Institut für Deutsche Sprache
kupietz@ids-mannheim.de

Harald Lüngen
Leibniz-Institut für Deutsche Sprache
luengen@ids-mannheim.de