Pipeline Effectiveness in the Sketch Engine

Matúš Kostka

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00, Brno, Czech Republic
xkostka4@fi.muni.cz

Abstract. This paper focuses on measuring the effectiveness of the most used language pipelines (48 pipelines) in Sketch Engine for potential future efficiency improvements. This paper will describe the tool for parallel measuring made for this task, analyze the problem before measuring, analyze and represent results from measured data, and end with a conclusion.

Keywords: Pipelines, Tokens, Sketch Engine, Effectiveness, Language

1 Introduction

Sketch Engine supports approximately 100 languages which, of course, follows a similar number of pipelines [1]. Pipeline stands for a group of tools for text processing, like normalization, tokenization/segmentation, lemmatization and part-of-speech tagger, in one executable file. It is necessary for corpus creation. Every language has at least one pipeline. Because Sketch Engine uses several types of pipelines, most of them are by the Sketch Engine team. However, some came from different creators, meaning pipelines can differ in features, way of functioning, efficiency, CPU, and RAM consumption. Nevertheless, the amount of attention paid to the pipeline often depends on the size and usage of language, the pipeline was created for[1].

The effectiveness of pipelines is crucial for processing vast amounts of data and also for user satisfaction while creating their own corpora with Sketch Engine. So this paper focuses on the measurement of pipeline effectiveness in Sketch Engine to create an overview of the actual state of pipelines for potential future improvements. The measured parameters are **total execution time, CPU usage** and **memory usage**. The paper will briefly describe the problem of the measurements, the tool created for this task, selected pipelines, analyze of measured data and ends with a conclusion.

2 Closer description

The goal is to create a universal tool and statistics for better orientation in the efficiency of the pipelines in Sketch Engine. The measured parameters are

[1] For a full list of pipelines features, visit https://www.sketchengine.eu/corpora-and-languages.

A. Horák, P. Rychlý, A. Rambousek (eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2022, pp. 123–130, 2022. © Tribun EU 2022

execution time, **CPU usage** and **memory usage** (maximum resident set size). It was realized on files of a specified amount of tokens, in this case, **10,000**; **100,000** and **1,000,000** tokens for every file. These numbers were selected because Sketch Engine enables users to create their own corpora but with a default upper limit of 1 million tokens [1]. Measurement by the number of tokens and not by the file size was decided because each language has a different length of words. This means if the measurement were done by the file size, the number of tokens would not be the same for every language. The specified tokens files are made chiefly from data downloaded from Wikipedia for individual languages in 2020 and 2021 by a web crawler SpiderLing [2]. The measurement took place at one of the servers of Lexical Computing CZ, with 32 cores and 256 GB of ram. The result can be influenced by the background processes, the pipeline version, and the content of the used files on the server.

2.1 The tool

The script for this task is written in bash and can create a file of a given amount of tokens, measure execution time, CPU and RAM usage, and export measured data in CVS format for future processing. The script is based on UNIX command **/usr/bin/time**, which measures already mentioned parameters [8]. Creating a file with the requested amount of tokens is quite a time-consuming process because the data are first decompressed from gunzip format, tokenized, then the tokens are counted for a specified amount and turned back via **vert2plain** function to prevertical form. The script can work in 2 modes: creating a temporary file or with an already created file, to save time while repeating the measurement more time. It is recommended to run the script via makefile with **-j**, which will run several jobs simultaneously. The only limitation here is only the number of CPUs and cores the machine offers.

2.2 Used pipelines

Totally 48 pipelines are measured. Some languages are measured more times like Italian, French, Greek because they use more versions of pipelines, but on the other side traditional and simplified Chinese uses the same pipeline. These 48 pipelines are the most used pipelines in Sketch Engine and that is the reason why they were selected. There are several aspects causing that these pipelines can differ in results like the number of supporting features, way of implementation, type of alphabet, and unique language characteristics. When the pipeline support all features (mostly tagging and lemming) huge language models are loaded in the initialization phase, which can be time, CPU and RAM consuming. Model loading is crucial in pipelines for languages with a unique alphabet, like Chinese, Japanese, Arabian, Bulgarian and for languages similar to these. Bear in mind that quicker pipelines in the result can support fewer. In table 1, See Table 1, can be seen pipeline features with quick description.
Notes to Table 1, See Table 1:

Table 1: Pipeline composition

Feature name	Description
Uninorm	Normalization of text convert the content into NFKC normalization form. [5]
Unitok	Tokenisation of a text is the process of splitting the text into units suitable for further computational processing. It is an important data preparation step allowing to perform more advanced tasks. [4]
Lemmatizer	Lemmatization is a process of assigning a lemma to each word form in a corpus. [6]
Treetagger	Assigning special labels to each token in the corpus to indicate part of speech, grammatical categories. [7]

Note that features can differ for each pipeline. Like tokenization for Chinese, Japanese is called segmentation.

3 Analyze

It is evident that the amount of resources used is directly proportional to the number of tokens. Closer result from measurement with a million tokens can be seen in [1, 2, 3].

Table 2: Stats 10,000 tokens

	Min value	Max value	Average	Median
Execution time (sec)	2.47	762.7	78.27	54.46
CPU usage (%)	0	100	26	18
RAM usage (GB)	0.007	2.326	0.252	0.141

Notes to Table 2, See Table 2:
In the row of execution time, the minimum value was reached by **Hebrew pipeline** and the maximum value by **Tagalog pipeline**. In the CPU usage row, the minimum was also reached by **Hebrew pipeline** but the maximum by **Japanese pipeline**. And from the RAM point of view, the minimum of it was used by **Thai pipeline** and the maximum again by **Tagalog pipeline**.
Notes to Table 3, See Table 3:
The fastest execution time had **universal pipeline**, a default pipeline for languages without their pipeline. It supports just normalization and tokenization. The slowest was again **Tagalog pipeline**. From the CPU point of view, the least CPU resources were used by **Hebrew pipeline** and the most by **Japanese pipeline**. More than 100% of CPU usage is possible because the program is

Table 3: Stats 100,000 tokens

	Min value	Max value	Average	Median
Execution time (sec)	4.21	3300.14	271.02	108.76
CPU usage (%)	0	127	40	38
RAM usage (GB)	0.008	5.443	0.388	0.187

run on more cores (multiprocessing), 1 core == 100%. The maximum possible is 3200% because the server on which the measurement was realized have 32 cores. For more information, See `https://en.wikipedia.org/wiki/Multiprocessing`. And from the RAM point of view, the minimum of it was used by **Thai pipeline** and the maximum again by **Tagalog pipeline**.

Table 4: Stats 1,000,000 tokens

	Min value	Max value	Average	Median
Execution time (sec)	23.7	8109.08	1020.49	395.78
CPU usage (%)	0	222	75	77
RAM usage (GB)	0.008	5.629	0.733	0.209

Notes to Table 4, See Table 4:
The results are again similar as the previous two measurements and the fastest execution time was reached by **Universal pipeline** and the slowest by **Hebrew pipeline** (different pipeline as previous). The minimum of CPU resources was used by **Thai pipeline** and the most by **Italian pipeline**. And the RAM usage, minimum of it, was used by **Thai pipeline** and the most by **Japanese pipeline**.

4 Conclusions

The Tagalog pipeline has been the worst from the analysis of all three types of measurements. Even at the 1 million tokens measurement, it did not finish at all, the pipeline was repeatedly restarting itself. Also, the low usage of RAM and CPU by **Thai pipeline** (thai_sw1) and **Hebrew pipeline** (yap_he_v1) is questionable and the most probably not alright, but all repetitions of the measurements show similar values.

It is clear that pipelines for languages with different alphabets as Latin are usually slower. Also, the fact that 46% of pipelines are slower than 10 minutes is alarming and requires some attention [1]. The RAM usage is all right, only two pipelines use more than 5 GB, and those are tagalog_sw1 and mecab_unidic_comainu_jpn [2]. Moreover, from the CPU point of view, its evident that only 12% of pipelines are multithreaded [3].

One Positive fact is that all pipelines are in a linear relationship with the number of tokens. Hence, it is possible to calculate linear regression for quicker

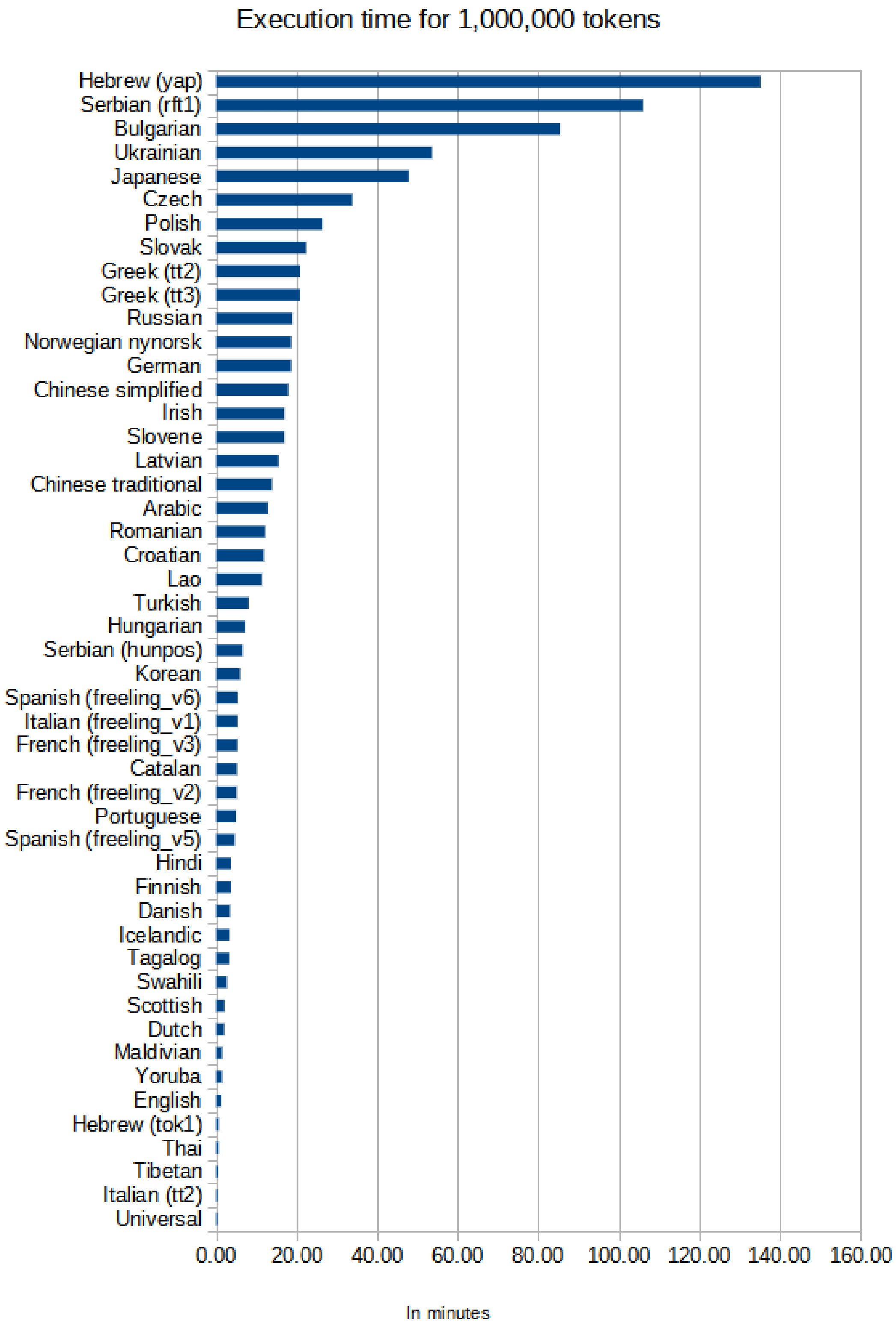

Fig. 1: Execution time for measured pipelines

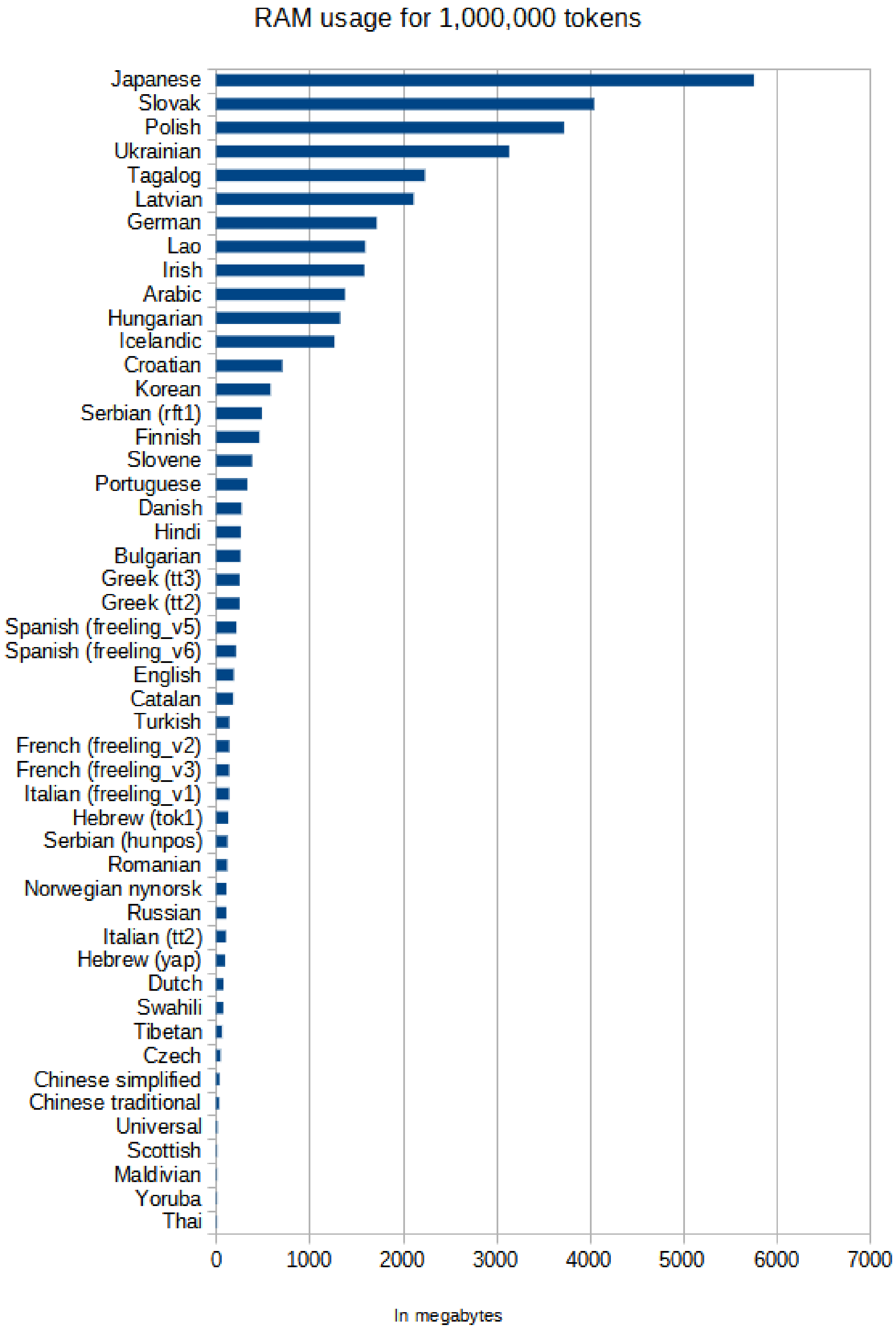

Fig. 2: RAM usage for measured pipelines

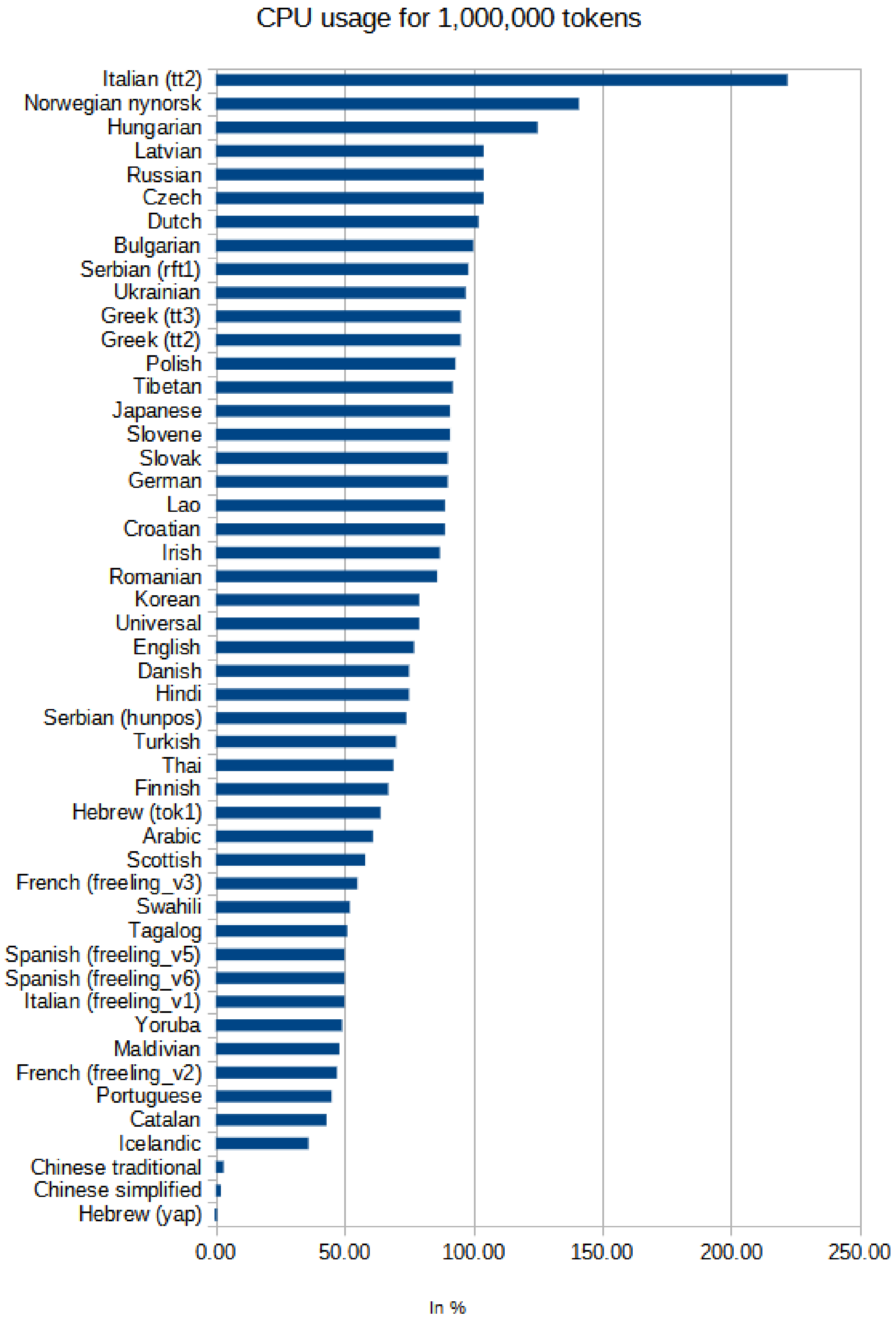

Fig. 3: CPU usage for measured pipelines

execution time estimation for a given token amount. However, it requires more measurements with a different number of tokens for a more precise result.

5 Possible future improvements and goals

The good idea is to realize more measurements with different amounts of tokens to have enough data to count linear regression with an acceptable result. Because now, there are just three measurements per pipeline from which the linear regression can be calculated but the error rate is quite high. The goal is to have at least ten measurements of different token numbers per pipeline. The next goal could be dividing pipelines according to supported features and measuring pipelines with similar features together because now it is highly probable that the fastest pipeline supports fewer features than the slower ones. And as the last goal and probably the most useful would be to measure the actual pipeline for all languages accessible in Sketch Engine.

Acknowledgments This publication has been fully supported by the Lexical Computing CZ s.r.o.

References

1. Kilgarriff, A., Baisa, V., Bušta, J., Jakubíček, M., Kovář, V., Michelfeit, J., Rychlý, P., Suchomel, V.: The sketch Engine: ten years on. Lexicography **1**(1), 17-19; 26-29 (2014)
2. Suchomel, V., Pomikálek, J.: Efficient web crawling for large text corpora. In: Proceedings of the seventh Web as Corpus Workshop (WAC7). (2012) 39-43
3. Languages in Sketch Engine, `https://www.sketchengine.eu/corpora-and-languages/`. Last accessed 16 Nov 2022
4. Michelfeit, J., Pomikálek, J., Suchomel, V.: Text Tokenisation Using unitok. (2014)
5. Unicode Normalization Forms, `https://unicode.org/reports/tr15/`. Last accesed 20 Nov 2022
6. Lemmatization, `https://www.sketchengine.eu/my_keywords/lemmatization/`. Last accesed 20 Nov 2022
7. POS tags, `https://www.sketchengine.eu/blog/pos-tags/`. Last accesed 20 Nov 2022
8. Time(1) — Linux manual page, `https://man7.org/linux/man-pages/man1/time.1.html`. Last accesed 20 Nov 2022