Sorbonne Université Université

Ecole Doctorale Informatique, Télécommunications et Electronique - ED130

Inria de Paris / Équipe ALMAnaCH

Thèse de doctorat
Discipline : Informatique

Présentée par

Pedro Ortiz Suarez

Dirigée par

Laurent Romary et Benoît Sagot

Pour obtenir le grade universitaire de

Docteur de Sorbonne Université

On Language Modeling and its Applications for Contemporary and Historical French

---

Présentée et soutenue publiquement le 30 avril 2022 devant le jury composé de :

Alonzo Church	Princeton University	Examinateur
Christopher Columbus	Kingdom of Castile	Invited member
Margaret Hamilton	University of Michigan	Rapporteur
Emmy Noether	Georg-August-Universität Göttingen	Examinateur
Laurent Romary	Inria - ALMAnaCH	Directeur
Benoît Sagot	Inria - ALMAnaCH	Co-directeur
Claude Shannon	MIT	Examinateur
Alan Turing	Princeton University	Rapporteur

Abstract

Scientific documents often use LaTeX for typesetting. While numerous packages and templates exist, it makes sense to create a new one. Just because.

Contents

Contents

1 INTRODUCTION

> In which the reasons for doing this Ph.D. are laid bare for the whole world to see and we encounter some answers to questions in which, frankly, only an extremely small number of people were interested in the first place.

This package contains a minimal, modern template for writing your thesis. While originally meant to be used for a Ph. D. thesis, you can equally well use it for your honour thesis, bachelor thesis, and so on—some adjustments may be necessary, though.

1.1 WHY?

I was not satisfied with the available templates for LaTeX and wanted to heed the style advice given by people such as Robert Bringhurst or Edward R. Tufte. While there *are* some packages out there that attempt to emulate these styles, I found them to be either too bloated, too playful, or too constraining. This template attempts to produce a beautiful look without having to resort to any sort of hacks. I hope you like it.

1.2 HOW?

The package tries to be easy to use. If you are satisfied with the default settings, just add

```
\documentclass{mimosis}
```

at the beginning of your document. This is sufficient to use the class. It is possible to build your document using either LaTeX|, X∃LaTeX, or LuaLaTeX. I personally prefer one of the latter two because they make it easier to select proper fonts.

Package	Purpose
amsmath	Basic mathematical typography
amsthm	Basic mathematical environments for proofs etc.
booktabs	Typographically light rules for tables
bookmarks	Bookmarks in the resulting PDF
dsfont	Double-stroke font for mathematical concepts
graphicx	Graphics
hyperref	Hyperlinks
multirow	Permits table content to span multiple rows or columns
paralist	Paragraph ('in-line') lists and compact enumerations
scrlayer-scrpage	Page headings
setspace	Line spacing
siunitx	Proper typesetting of units
subcaption	Proper sub-captions for figures

Table 1.1: A list of the most relevant packages required (and automatically imported) by this template.

1.3 FEATURES

The template automatically imports numerous convenience packages that aid in your typesetting process. Table 1.1 lists the most important ones. Let's briefly discuss some examples below. Please refer to the source code for more demonstrations.

1.3.1 TYPESETTING MATHEMATICS

This template uses amsmath and amssymb, which are the de-facto standard for typesetting mathematics. Use numbered equations using the equation environment. If you want to show multiple equations and align them, use the align environment:

$$V := \{1, 2, \dots\} \tag{1.1}$$

$$E := \{(u, v) \mid \operatorname{dist}(p_u, p_v) \leq \epsilon\} \tag{1.2}$$

Define new mathematical operators using \DeclareMathOperator. Some operators are already pre-defined by the template, such as the distance between two objects. Please see the template for some examples. Moreover, this template contains a correct differential operator. Use \diff to typeset the differential of integrals:

$$f(u) := \int_{v \in \mathbb{D}} \operatorname{dist}(u, v) \, \mathrm{d}v \tag{1.3}$$

You can see that, as a courtesy towards most mathematicians, this template gives you the possibility to refer to the real numbers $\mathbb{R}$ and the domain $\mathbb{D}$ of some function.

Take a look at the source for more examples. By the way, the template comes with spacing fixes for the automated placement of brackets.

1.3.2 TYPESETTING TEXT

Along with the standard environments, this template offers `paralist` for lists within paragraphs. Here's a quick example: The American constitution speaks, among others, of (i) life (ii) liberty (iii) the pursuit of happiness. These should be added in equal measure to your own conduct. To typeset units correctly, use the `siunitx` package. For example, you might want to restrict your daily intake of liberty to 750 mg.

Likewise, as a small pet peeve of mine, I offer specific operators for *ordinals*. Use \th to typeset things like July 4$^{\text{th}}$ correctly. Or, if you are referring to the 2$^{\text{nd}}$ edition of a book, please use \nd. Likewise, if you came in 3$^{\text{rd}}$ in a marathon, use \rd. This is my 1$^{\text{st}}$ rule.

1.4 CHANGING THINGS

Since this class heavily relies on the `scrbook` class, you can use *their* styling commands in order to change the look of things. For example, if you want to change the text in sections to **bold** you can just use

```
\setkomafont{sectioning}{\normalfont\bfseries}
```

at the end of the document preamble—you don't have to modify the class file for this. Please consult the source code for more information.

Part I

Data

2   OSCAR

2.1 GOCLASSY

2.2 Introduction

In recent years neural methods for Natural Language Processing (NLP) have consistently and repeatedly improved the state-of-the-art in a wide variety of NLP tasks such as parsing, PoS-tagging, named entity recognition, machine translation, text classification and reading comprehension among others. Probably the main contributing factor in this steady improvement for NLP models is the raise in usage of *transfer learning* techniques in the field. These methods normally consist of taking a pre-trained model and reusing it, with little to no retraining, to solve a different task from the original one it was intended to solve; in other words, one *transfers* the *knowledge* from one task to another.

Most of the transfer learning done in NLP nowadays is done in an unsupervised manner, that is, it normally consist of a *language model* that is fed unannotated plain text in a particular language; so that it *extracts* or *learns* the basic *features* and patterns of the given language, the model is subsequently used on top of an specialised architecture designed to tackle a particular NLP task. Probably the best known example of this type of model are *word embeddings* which consist of real-valued vector representations that are trained for each word on a given corpus. Some notorious examples of word embeddings are word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) and fastText (Mikolov et al., 2018). All these models are *context-free*, meaning that a given word has one single vector representation that is independent of context, thus for a polysemous word like Washington, one would have one single representation that is reused for the city, the state and the US president.

In order to overcome the problem of polysemy, *contextual* models have recently appeared. Most notably ELMo (Peters et al., 2018) which produces deep contextualised word representations out of the internal states of a deep bidirectional language model in order to model word use and how the usage varies across linguistic contexts. ELMo still needs to be used alongside a specialised architecture for each given downstream task, but newer architectures that can be fine-tuned have also appear. For these, the model is first fed unannotated data, and is then fine-tuned with annotated data to a particular downstream task without relying on any other architecture. The most remarkable examples of this type of model are GPT-1, GPT-2 (Radford et al., 2018, 2019), BERT (Devlin et al., 2019) and XLNet (Yang et al., 2019); the latter being the current state-of-the-art for multiple downstream tasks. All of these models are different arrangements of the Transformer architecture (Vaswani et al., 2017) trained with different datasets, except for XLNet which is an instance of the Transformer-XL (Dai et al., 2019).

Even though these models have clear advantages, their main drawback is the amount of data that is needed to train them in order to obtain a functional and efficient model. For the first English version of word2vec, Mikolov et al. (2013) used a one billion word dataset consisting of various news articles. Later Al-Rfou' et al. (2013) and then Bojanowski et al. (2017) used the plain text from Wikipedia to

train distributions of word2vec and fastText respectively, for languages other than English. Now, the problem of obtaining large quantities of data aggravates even more for contextual models, as they normally need multiple instances of a given word in order to capture all its different uses and in order to avoid overfitting due to the large quantity of hyperparameters that these models have. Peters et al. (2018) for example use a 5.5 billion token[1] dataset comprised of crawled news articles plus the English Wikipedia in order to train ELMo, Devlin et al. (2019) use a 3.3 billion word[2] corpus made by merging the English Wikipedia with the BooksCorpus (Zhu et al., 2015), and Radford et al. (2019) use a 40GB English corpus created by scraping outbound links from Reddit.[3]

While Wikipedia is freely available, and multiple pipelines exist[4,5] to extract plain text from it, some of the bigger corpora mentioned above are not made available by the authors either due to copyright issues or probably because of the infrastructure needed to serve and distribute such big corpora. Moreover the vast majority of both these models and the corpora they are trained with are in English, meaning that the availability of high quality NLP for other languages, specially for low-resource languages, is rather limited.

To address this problem, we choose Common Crawl[6], which is a 20TB mutilingual free to use corpus composed of crawled websites from the internet, and we propose a highly parallel multithreaded asynchronous pipeline that applies well-known concurrency patterns, to clean and classify by language the whole Common Crawl corpus to a point where it is usable for Machine Learning and in particular for neural NLP applications. We optimise the pipeline so that the process can be completed in a sensible amount of time even in infrastructures where Input/Output (I/O) speeds become the main bottleneck.

Knowing that even running our pipeline will not always be feasible, we also commit to publishing our own version of a classified by language, filtered and ready to use Common Crawl corpus upon publication of this article. We will set up an easy to use interface so that people can download a manageable amount of data on a desired target language.

2.3 Related Work

Common Crawl has already been successfully used to train language models, even multilingual ones. The most notable example in probably fastText which was first trained for English using Common Crawl (Mikolov et al., 2018) and then for other

---

[1]Punctuation marks are counted as tokens.
[2]Space sparated tokens.
[3]https://www.reddit.com/
[4]https://github.com/attardi/wikiextractor
[5]https://github.com/hghodrati/wikifil
[6]http://commoncrawl.org/

157 different languages (Grave et al., 2018). In fact Grave et al. (2018) proposed a pipeline to filter, clean and classify Common Crawl, which we shall call the "fast-Text pre-processing pipeline." They used the fastText linear classifier (Joulin et al., 2016, 2017) to classify each line of Common Crawl by language, and downloaded the initial corpus and schedule the I/O using some simple Bash scripts. Their solution, however, proved to be a synchronous blocking pipeline that works well on infrastructures having the necessary hardware to assure high I/O speeds even when storing tens of terabytes of data at a time. But that downscales poorly to medium-low resource infrastructures that rely on more traditional cost-effective electromechanical mediums in order to store this amount of data.

Concerning contextual models, Baevski et al. (2019) trained a BERT-like bi-directional Transformer for English using Common Crawl. They followed the "fastText pre-processing pipeline" but they removed all copies of Wikipedia inside Common Crawl. They also trained their model using News Crawl (Bojar et al., 2018) and using Wikipedia + BooksCorpus, they compared three models and showed that Common Crawl gives the best performance out of the three corpora.

The XLNet model was trained for English by joining the BookCorpus, English Wikipedia, Giga5 (Parker et al., 2011), ClueWeb 2012-B (Callan et al., 2009) and Common Crawl. Particularly for Common Crawl, Yang et al. (2019) say they use "heuristics to aggressively filter out short or low-quality articles" from Common Crawl, however they don't give any detail about these "heuristics" nor about the pipeline they use to classify and extract the English part of Common Crawl.

It is important to note that none of these projects distributed their classified, filtered and cleaned versions of Common Crawl, making it difficult in general to faithfully reproduce their results.

2.4 COMMON CRAWL

Common Crawl is a non-profit foundation which produces and maintains an open repository of web crawled data that is both accessible and analysable.[7] Common Crawl's complete web archive consists of petabytes of data collected over 8 years of web crawling. The repository contains raw web page HTML data (WARC files), metdata extracts (WAT files) and plain text extracts (WET files). The organisation's crawlers has always respected `nofollow`[8] and `robots.txt`[9] policies.

Each monthly Common Crawl snapshot is in itself a massive multilingual corpus, where every single file contains data coming from multiple web pages written in a large variety of languages and covering all possible types of topics. Thus, in order to effectively use this corpus for the previously mentioned Natural Language Pro-
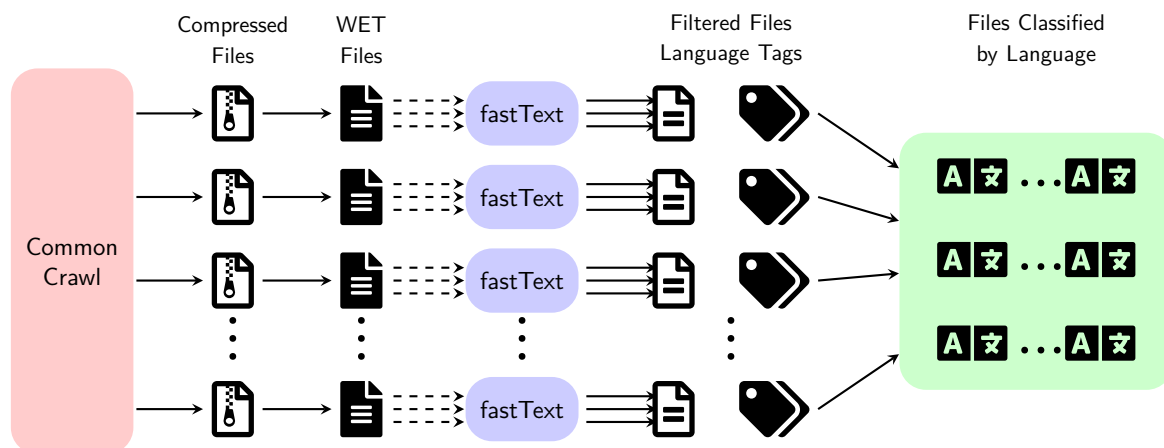
---

[7] http://commoncrawl.org/about/
[8] http://microformats.org/wiki/rel-nofollow
[9] https://www.robotstxt.org/

Figure 2.1: A scheme of the *goclassy* pipeline. The red square represents the Compressed WET files stored on Amazon Web Services. The 🗄 icons represent the gzip files stored locally, the 📄 represent one of the 50K WET files. The 📄 represents the filtered file and the 🏷 represents a file of language tags, one tag per line in 📄. The 🆎 represents one of the 166 classified files. Each arrow represents an asynchronous non blocking worker and dotted arrows represent a line filtering process.

cessing and Machine Learning applications, one has first to extract, filter, clean and classify the data in the snapshot by language.

For our purposes we use the WET files which contain the extracted plain texts from the websites mostly converted to UTF-8, as well as headers containing the metatada of each crawled document. Each WET file comes compressed in gzip format[10] and is stored on Amazon Web Services. We use the November 2018 snapshot which surpasses 20TB of uncompressed data and contains more than 50 thousand plain text files where each file consists of the plain text from multiple websites along its metadata header. From now on, when we mention the "Common Crawl" corpus, we refer to this particular November 2018 snapshot.

2.5 FASTTEXT'S PIPELINE

In order to download, extract, filter, clean and classify Common Crawl we base ourselves on the "fastText pre-processing pipeline" used by Grave et al. (2018). Their pipeline first launches multiple process, preferably as many as available cores. Each of these processes first downloads one Common Crawl WET file which then proceeds to decompress after the download is over. After decompressing, an instance of the fastText linear classifier (Joulin et al., 2016, 2017) is launched, the classifier processes each WET file line by line, generating a language tag for each line. The

---

[10]https://www.gnu.org/software/gzip/

tags are then stored in a tag file which holds a one-to-one correspondence between lines of the WET file and its corresponding language tag. The WET file and the tag files are read sequentially and each on the WET file line holding the condition of being longer that 100 bytes is appended to a language file containing only plain text (tags are discarded). Finally the tag file and the WET files are deleted.

Only when one of these processes finishes another can be launched. This means that one can at most process and download as many files as cores the machine has. That is, if for example a machine has 24 cores, only 24 WET files can be downloaded and processed simultaneously, moreover, the 25th file won't be downloaded until one of the previous 24 files is completely processed.

When all the WET files are classified, one would normally get around 160 language files, each file holding just plain text written in its corresponding language. These files still need to be filtered in order to get rid of all files containing invalid UTF-8 characters, so again a number of processes are launched, this time depending on the amount of memory of the machine. Each process reads a language file, first filters for invalid UTF-8 characters and then performs deduplication. A simple non-collision resistant hashing algorithm is used to deduplicate the files.

The fastText linear classifier works by representing sentences for classification as Bags of Words (BoW) and training a linear classifier. A weight matrix A is used as a look-up table over the words and the word representations are then averaged into a text representation which is fed to the linear classifier. The architecture is in general similar to the CBoW model of Mikolov et al. (2013) but the middle word is replaced by a label. They uses a softmax function f to compute the probability distribution over the classes. For a set of N documents, the model is trained to minimise the negative log-likelihood over the classes:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)),$$

where x_n is the normalised bag of features of the n-th document, y_n is the n-th label, and A, B are the weight matrices. The pre-trained fastText model for language recognition (Grave et al., 2018) is capable of recognising around 176 different languages and was trained using 400 million tokens from Wikipedia as well as sentences from the Tatoeba website[11].

2.6 ASYNCHRONOUS PIPELINE

We propose a new pipeline derived from the fastText one which we call `goclassy`, we reuse the fastText linear classifier (Joulin et al., 2016, 2017) and the pre-trained

---

[11]https://tatoeba.org/

	10 files			100 files			200 files		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
real									
fastText	2m50s	6m45s	3m31s	13m46s	38m38s	17m39s	26m20s	47m48s	31m4s
goclassy	1m23s	3m12s	1m42s	7m42s	12m43s	9m8s	15m3s	15m47s	15m16s
user									
fastText	26m45s	27m2s	26m53s	4h21m	4h24m	4h23m	8h42m	8h48m	8h45m
goclassy	10m26s	12m53s	11m0s	1h46m	1h54m	1h49m	3h37m	3h40m	3h38m
sys									
fastText	40.14s	40.85s	40.56s	6m14s	6m17s	6m15s	12m26s	12m45s	12m31s
goclassy	37.34s	45.98s	39.67s	5m7s	5m34s	5m16s	9m57s	10m14s	10m5s

Table 2.1: Benchmarks are done using the UNIX `time` tool, are repeated 10 times each and are done for random samples of 10, 100 and 200 WET files. Only the classifying and filtering part are benchmarked. The table shows the minimum, maximum and mean time for the user, real and sys time over the 10 runs. Here "fastText" is used as short for the pipeline.

fastText model for language recognition (Grave et al., 2018), but we completely rewrite and parallelise their pipeline in an asynchronous manner.

The order of operations is more or less the same as in the fastText pre-processing pipeline but instead of clustering multiple operations into a single blocking process, we launch a worker for each operation and we bound the number of possible parallel operations at a given time by the number of available threads instead of the number of CPUs. We implement goclassy using the Go programming language[12] so we let the Go runtime[13] handle the scheduling of the processes. Thus in our pipeline we don't have to wait for a whole WET file to download, decompress and classify in order to start downloading and processing the next one, a new file will start downloading and processing as soon as the scheduler is able to allocate a new process.

When using electromechanical mediums of storage, I/O blocking is one of the main problems one encounters. To overcome this, we introduced buffers in all our I/O operations, a feature that is not present in the fastText pre-processing pipeline. We also create, from the start, a file for each of the 176 languages that the pre-trained fastText language classifier is capable of recognising, and we always leave them open, as we find that getting a file descriptor to each time we want to write, if we wanted leave them open just when needed, introduces a big overhead.

We also do the filtering and cleaning processes at line level before feeding each line to the classifier, which makes us create a new filtered file so that we can have a correspondence with the tag file, which in turn will consume more space, but that will also reduce the amount of unnecessary classifications performed by fastText.

---

[12]https://golang.org/
[13]https://golang.org/src/runtime/mprof.go

The filtered and file tags are then read and lines are appended to its corresponding language file. The writing in the classification step is asynchronous, meaning that process writing a line to the filtered files does not wait for the classifier to write a tag on the tag file. Figure 2.1 shows the pipeline up to this point.

After all WET files are processed, we then use Isaac Whitfield's deduplication tool runiq[14] which is based on Yann Collet's xxhash64[15], an extremely fast non-cryptographic hash algorithm that is resistant to collisions. We finally use the Mark Adler's pigz[16] for data compression, as opposed to the canonical UNIX tools proposed in the original fastText pipeline. We add both tools to our concurrent pipeline, executing multiple instances of them in parallel, in order to ensure we use the most of our available resources at a given time.

Beyond improving the computational time required to classify this corpus, we propose a simple improvement on the cleaning scheme in the fastText pre-processing pipeline. This improvement allows our pipeline to better take into account the multilingual nature of Common Crawl; that is, we count UTF-8 characters instead of bytes for setting the lower admissible bound for the length of a line to be fed into the classifier. This straightforward modification on the fastText pre-processing pipeline assures we take into account the multiple languages present in Common Crawl that use non-ASCII encoded characters.

Given that our implementation is written in Go, we release binary distributions [17] of goclassy for all major operating systems. Both pigz and runiq are also available for all major operating systems.

2.7 BENCHMARKS

We test both pipelines against one another in an infrastructure using traditional electromechanical storage mediums that are connected to the main processing machine via an Ethernet interface, that is, a low I/O speed environment as compared to an infrastructure where one would have an array of SSDs connected directly to the main processing machine via a high speed interface. We use a machine with an Intel® Xeon® Processor E5-2650 2.00 GHz, 20M Cache, and 203.1 GiB of RAM. We make sure that no other processes apart from the benchmark and the Linux system processes are run. We do not include downloading, decompression or deduplication in our benchmarks as downloading takes far too much time, and deduplication and compression were performed with third party tools that don't make part of our main contribution. We are mainly interested in seeing how the way the data is fed to the classifier impacts the overall processing time.

---

[14] https://github.com/whitfin/runiq
[15] https://github.com/Cyan4973/xxHash
[16] https://zlib.net/pigz/
[17] https://github.com/pjox/goclassy

Benchmarks in table 2.1 of our goclassy pipeline show a drastic reduction in processing time compared to the original fastText prepossessing pipeline. We show that in our particular infrastructure, we are capable of reducing the *real* time as measured by the `time` UNIX tool almost always by half. The *user* time which represents the amount of CPU time spent in user-mode code (outside the kernel) within the process is almost three times lower for our goclassy pipeline, this particular benchmark strongly suggest a substantial reduction in energy consumption of goclassy with respect to the fastText pipeline.

As we understand that even an infrastructure with more than 20TB of free space in traditional electromechanical storage is not available to everyone and we propose a simple parametrization in our pipeline that actively deletes already processed data and that only downloads and decompresses files when needed, thus ensuring that no more than 10TB of storage are used at a given time. We nevertheless note that delaying decompression increases the amount of computation time, which is a trade-off that some users might make as it might be more suitable for their available infrastructure.

2.8 OSCAR

Finally, we are aware that some users might not even have access to a big enough infrastructure to run our pipelines or just to store all the Common Crawl data. Moreover, even if previously used and cited in NLP and Machine Learning research, we note that there is currently no public distribution of Common Crawl that is filtered, classified by language and ready to use for Machine Learning or NLP applications. Thus we decide to publish a pre-processed version of the November 2018 copy of Common Crawl which is comprised of usable data in 166 different languages, we publish[18] our version under the name OSCAR which is short for *Open Super-large Crawled ALMAnaCH[19] coRpus*.

After processing all the data with goclassy, the size of the whole Common Crawl corpus is reduced to 6.3TB, but in spite of this considerable reduction, OSCAR still dwarfs all previous mentioned corpora having more 800 billion "words" or spaced separated tokens and noting that this in fact is an understatement of how big OSCAR is, as some of the largest languages within OSCAR such as Chinese and Japanese do not use spaces. The sizes in bytes for both the original and the deduplicated versions of OSCAR can be found in table 2.2. OSCAR is published under the *Creative Commons CC0 license* (*"no rights reserved"*)[20], so it is free to use for all applications.

---

[18]https://team.inria.fr/almanach/oscar/
[19]https://team.inria.fr/almanach/
[20]http://creativecommons.org/publicdomain/zero/1.0/

2.9 Conclusions

We are sure that our work will greatly benefit researchers working on an either constrain infrastructure or a low budget setting. We are also confident, that by publishing a classified version of Common Crawl, we will substantially increase the amount of available public data for medium to low resource languages, thus improving and facilitating NLP research for them. Furthermore, as our pipeline speeds-up and simplifies the treatment of Common Crawl, we believe that our contribution can be further parallelised and adapted to treat multiple snapshots of Common Crawl opening the door to what would be otherwise costly diachronic studies of the use of a given language throughout the internet.

Finally, we note that both our proposed pipeline is data independent, which means that they can be reused to process, clean and classify any sort of big multilingual corpus that is available in plain text form and that is UTF-8 encoded; meaning that the impact of our work goes way beyond a single corpus.

Language	Size		Words	
	Orig	Dedup	Orig	Dedup
Afrikaans	241M	163M	43,482,801	29,533,437
Albanian	2.3G	1.2G	374,196,110	186,856,699
Amharic	360M	206M	28,301,601	16,086,628
Arabic	82G	32G	8,117,162,828	3,171,221,354
Aragonese	1.3M	801K	52,896	45,669
Armenian	3.7G	1.5G	273,919,388	110,196,043
Assamese	113M	71M	6,956,663	4,366,570
Asturian	2.4M	2.0M	381,005	325,237
Avaric	409K	324K	24,720	19,478
Azerbaijani	2.8G	1.5G	322,641,710	167,742,296
Bashkir	128M	90M	9,796,764	6,922,589
Basque	848M	342M	120,456,652	45,359,710
Bavarian	503	503	399	399
Belarusian	1.8G	1.1G	144,579,630	83,499,037
Bengali	11G	5.8G	623,575,733	363,766,143
Bihari	110K	34K	8,848	2,875
Bishnupriya	4.1M	1.7M	198,286	96,940
Bosnian	447K	116K	106,448	20,485
Breton	29M	16M	5,013,241	2,890,384
Bulgarian	32G	14G	2,947,648,106	1,268,114,977
Burmese	1.9G	1.1G	56,111,184	30,102,173
Catalan	8.0G	4.3G	1,360,212,450	729,333,440
Cebuano	39M	24M	6,603,567	3,675,024
Central Bikol	885	885	312	312
Central Khmer	1.1G	581M	20,690,610	10,082,245
Central Kurdish	487M	226M	48,478,334	18,726,721
Chavacano	520	520	130	130
Chechen	8.3M	6.7M	711,051	568,146
Chinese	508G	249G	14,986,424,850	6,350,215,113
Chuvash	39M	26M	3,041,614	2,054,810
Cornish	44K	14K	8,329	2,704
Croatian	226M	110M	34,232,765	16,727,640
Czech	53G	24G	7,715,977,441	3,540,997,509
Danish	16G	9.5G	2,637,463,889	1,620,091,317
Dhivehi	126M	79M	7,559,472	4,726,660
Dimli	146	146	19	19
Dutch	78G	39G	13,020,136,373	6,598,786,137
Eastern Mari	7.2M	6.0M	565,992	469,297
Egyptian Arabic	66M	33M	7,305,151	3,659,419
Emilian-Romagnol	25K	24K	6,376	6,121
English	2.3T	1.2T	418,187,793,408	215,841,256,971
Erzya	1.4K	1.2K	90	78
Esperanto	299M	228M	48,486,161	37,324,446
Estonian	4.8G	2.3G	643,163,730	309,931,463
Finnish	27G	13G	3,196,666,419	1,597,855,468
French	282G	138G	46,896,036,417	23,206,776,649
Galician	620M	384M	102,011,291	63,600,602
Georgian	3.6G	1.9G	171,950,621	91,569,739
German	308G	145G	44,878,908,446	21,529,164,172
Goan Konkani	2.2M	1.8M	124,277	102,306
Guarani	36K	24K	7,382	4,680
Gujarati	1.1G	722M	72,045,701	50,023,432
Haitian	3.9K	3.3K	1,014	832
Hebrew	20G	9.8G	2,067,753,528	1,032,018,056
Hindi	17G	8.9G	1,372,234,782	745,774,934
Hungarian	40G	18G	5,163,936,345	2,339,127,555
Icelandic	1.5G	846M	219,900,094	129,818,331
Ido	147K	130K	25,702	22,773
Iloko	874K	636K	142,942	105,564
Indonesian	30G	16G	4,574,692,265	2,394,957,629
Interlingua	662M	360K	180,231	100,019
Interlingue	24K	1.6K	5,352	602
Irish	88M	60M	14,483,593	10,017,303
Italian	137G	69G	22,248,707,341	11,250,012,896
Japanese	216G	106G	4,962,979,182	1,123,067,063
Javanese	659K	583K	104,896	86,654
Kalmyk	113K	112K	10,277	10,155
Kannada	1.7G	1.1G	81,186,863	49,343,462
Karachay-Balkar	2.6M	2.3M	185,436	166,496
Kazakh	2.7G	1.5G	191,126,469	108,388,743
Kirghiz	600M	388M	44,194,823	28,982,620
Komi	2.3M	1.2M	201,404	95,243
Korean	24G	12G	2,368,765,142	1,120,375,149
Kurdish	94M	60M	15,561,003	9,946,440
Lao	174M	114M	4,133,311	2,583,342
Latin	26M	8.3M	4,122,201	1,328,038
Latvian	4.0G	1.8G	520,761,977	236,428,905
Lezghian	3.3M	3.0M	247,646	224,871
Limburgan	29K	27K	4,730	4,283
Lithuanian	8.8G	3.9G	1,159,661,742	516,183,525
Lojban	736K	678K	154,330	141,973
Lombard	443K	433K	75,229	73,665
Low German	18M	13M	2,906,347	2,146,417
Lower Sorbian	13K	7.1K	1,787	966
Luxembourgish	29M	21M	4,403,577	3,087,650
Macedonian	2.1G	1.2G	189,289,873	102,849,595
Maithili	317K	11K	69,161	874
Malagasy	21M	13M	3,068,360	1,872,044
Malay	111M	42M	16,696,882	6,045,753
Malayalam	4.9G	2.5G	189,534,472	95,892,551
Maltese	24M	17M	2,995,654	2,163,358
Marathi	2.7G	1.4G	162,609,404	82,130,803
Mazanderani	691K	602K	73,870	64,481
Minangkabau	608K	310K	5,682	4,825
Mingrelian	5.8M	4.4M	299,098	228,629
Mirandese	1.2K	1.1K	171	152
Modern Greek	62G	27G	5,479,180,137	2,412,419,435
Mongolian	2.2G	838M	181,307,167	68,362,013
Nahuatl languages	12K	11K	1,234	1,193
Neapolitan	17K	13K	5,282	4,147
Nepali	1.8G	1.2G	107,448,208	71,628,317
Newari	5.5M	4.1M	564,697	288,995
Northern Frisian	4.4K	4.4K	1,516	1,516
Northern Luri	76K	63K	8,022	6,740
Norwegian	8.0G	4.7G	1,344,326,388	804,894,377
Norwegian Nynorsk	85M	54M	14,764,980	9,435,139
Occitan	5.8M	3.7M	750,301	512,678
Oriya	248M	188M	14,938,567	11,321,740
Ossetian	13M	11M	1,031,268	878,765
Pampanga	760	304	130	130
Panjabi	763M	460M	61,847,806	37,555,835
Persian	79G	38G	9,096,554,121	4,363,505,319
Piemontese	2.1M	1.9M	362,013	337,246
Polish	109G	47G	15,277,255,137	6,708,709,674
Portuguese	124G	64G	20,641,903,898	10,751,156,918
Pushto	361M	242M	46,559,441	31,347,348
Quechua	78K	67K	10,186	8,691
Romanian	25G	11G	3,984,317,058	1,741,794,069
Romansh	7.4K	6.5K	1,093	960
Russia Buriat	13K	11K	963	809
Russian	1.2T	568G	92,522,407,837	46,692,691,520
Sanskrit	93M	37M	4,331,569	1,713,930
Scottish Gaelic	1.9M	1.3M	310,689	207,110
Serbian	3.9G	2.2G	364,395,411	207,561,168
Serbo-Croatian	25M	5.8M	5,292,184	1,040,573
Sicilian	3.3K	2.8K	554	468
Sindhi	347M	263M	43,530,158	33,028,015
Sinhala	1.4G	802M	93,053,465	50,864,857
Slovak	9.1G	4.5G	1,322,247,763	656,346,179
Slovenian	2.5G	1.3G	387,399,700	193,926,684
Somali	61M	16M	1,202	472
South Azerbaijani	27M	19M	2,175,054	1,528,709
Spanish	278G	149G	47,545,122,279	25,928,290,729
Sundanese	211M	141M	30,321	20,278
Swahili	13M	8.1M	2,211,927	1,376,963
Swedish	44G	25G	7,155,994,312	4,106,120,608
Tagalog	573M	407M	98,949,299	70,121,601
Tajik	379M	249M	31,758,142	21,029,893
Tamil	9.3G	5.1G	420,537,132	226,013,330
Tatar	670M	305M	51,034,893	23,825,695
Telugu	2.5G	1.6G	123,711,517	79,094,167
Thai	36G	16G	951,743,087	368,965,202
Tibetan	187M	138M	1,483,589	936,556
Tosk Albanian	5.0M	2.8M	841,750	459,001
Turkish	60G	27G	7,577,388,700	3,365,734,289
Turkmen	11M	6.8M	1,113,869	752,326
Tuvinian	12K	7.9K	759	540
Uighur	122M	83M	8,657,141	5,852,225
Ukrainian	53G	28G	4,204,381,276	2,252,380,351
Upper Sorbian	4.2M	1.8M	545,351	236,867
Urdu	2.7G	1.7G	331,817,982	218,030,228
Uzbek	21M	12M	2,450,256	1,381,644
Venetian	18K	17K	3,492	3,199
Vietnamese	68G	32G	12,036,845,359	5,577,159,843
Volapük	2.0M	2.0M	321,121	318,568
Walloon	273K	203K	50,720	37,543
Waray	2.5M	2.2M	397,315	336,311
Welsh	213M	133M	37,422,441	23,574,673
Western Frisian	35M	26M	5,691,077	4,223,816
Western Mari	1.2M	1.1M	93,338	87,780
Western Panjabi	12M	9.0M	1,426,986	1,111,112
Wu Chinese	109K	32K	11,189	4,333
Yakut	42M	26M	2,547,623	1,789,174
Yiddish	141M	84M	13,834,320	8,212,970
Yoruba	55K	27K	8,906	3,518
Yue Chinese	3.7K	2.2K	186	128
Total	6.3T	3.2T	844,315,434,723	425,651,344,234

Table 2.2: Size of the OSCAR corpus by language measured in bytes and number of words. Standard UNIX human-readable notation is used for the size in byte. We define "words" as spaced separated tokens, which gives a good estimate of the size of each corpus for languages using Latin or Cyrillic alphabets, but might give a misleading size for other languages such as Chinese or Japanese.

With the increasing interest in language modeling in recent years in Natural Language Processing (NLP) (Rogers et al., 2020), particularly concerning contextualized word representations[21] (Peters et al., 2018; Devlin et al., 2019), there has also been an explosion in interest for large raw corpora, as some of these latest models require almost 1TiB of raw text for pre-training (**??**).

While most of these language models were initially trained in English (Devlin et al., 2019; Yang et al., 2019; **?**; **?**; **?**) and consequently most of the large corpora used to pre-train them were in English, there has been a recent push to produce larger high quality corpora for other languages, namely those of Grave et al. (2018), CCNet (Wenzek et al., 2020), Multilingual C4 (mC4) (**?**) and OSCAR (Ortiz Suárez et al., 2019; Ortiz Suárez et al., 2020) for pre-training language models, as well as, Paracrawl (Esplà et al., 2019; Bañón et al., 2020), CCAligned (El-Kishky et al., 2020) and WikiMatrix (Schwenk et al., 2021) which are parallel corpora for training Machine Translation (MT) models. Of these, only OSCAR, Paracrawl, CCaligned and WikiMatrix are freely available and easily downloadable.

In this paper we propose a new multilingual corpus for language modeling, and for that we take inspiration in the OSCAR corpus and its pipeline goclassy[22] (Ortiz Suárez et al., 2019; Ortiz Suárez et al., 2020), but we propose a new pipeline *Ungoliant*[23] that is faster, modular, parametrizable and well-documented. We then use it to produce a new corpus similar to OSCAR, yet larger, based on recent data containing mentions of last years' events such as the COVID-19 pandemic, the 2020–2021 United States racial unrest, the Australian wildfires, the Beirut explosion and Brexit among others. Moreover, contrarily to OSCAR, our corpus retains metadata information at the document level. We release our pipeline under an Apache 2.0 open source license and we publish the corpus under a research-only use license following the licensing schemes proposed by OSCAR (Ortiz Suárez et al., 2019; Ortiz Suárez et al., 2020) and Paracrawl (Esplà et al., 2019; Bañón et al., 2020).

2.10 Limitations of the OSCAR Corpus and its Generation Pipeline

2.10.1 OSCAR

OSCAR is a multilingual corpus derived from CommonCrawl[24], a project that provides web crawl data for everyone on a periodic manner, usually each month. CommonCrawl provides data in several formats, from raw HTML source code to pure text. OSCAR was generated from the pure text data version (WET files) of the

---

[21]In which one takes a unnanotated large textual corpus in a particular language and tries to predict a missing word in order to learn a vector space representation for it.
[22]https://github.com/oscar-corpus/goclassy
[23]https://github.com/oscar-corpus/ungoliant
[24]https://commoncrawl.org

November 2018 crawl, distributed in the form of 56,000 *shards*, that were then filtered and classified by language (Ortiz Suárez et al., 2019; Ortiz Suárez et al., 2020). OSCAR is available through several means, and has been used in numerous projects (Ortiz Suárez et al., 2019). OSCAR's generation pipeline also suffers from numerous issues, which we plan to address simultaneously with the release of a new, more powerful, stable, and higher quality pipeline

Simply put, OSCAR is composed of single language files that contain textual data (`ta.txt` for the Tamil language, for example). However, due to the often huge sizes of these files, and subsequently the impracticality of storage and distribution, OSCAR files are split and compressed in equally sized parts.

OSCAR comes in four different versions, each suited differently for different tasks, and allows less limited ways of sharing the corpus more widely. These versions are either *unshuffled* or *shuffled* (that is, for each language, lines have been shuffled, destroying records integrity), and *non-deduplicated* or *deduplicated* (since duplicate lines account for more than half of the total data[25] generated by the pipeline). For the unshuffled versions, each language file contains paragraphs that come from the same record, and each paragraph is separated by a newline.

OSCAR is inherently linked to its generation pipeline, and as such its quality partly depends on the pipeline's quality. While OSCAR is considered to be one of the cleanest multilingual corpora available (Caswell et al., 2020; **?**), several problems have been described, and the state of the publicly available code raises questions about maintenance and maintenability of the pipeline itself.

Apart from the fact that its content dates back to 2018, the current OSCAR corpus suffers from quality issues discussed in (Caswell et al., 2020; **?**), including:

- **Language label mismatches and inconsistencies**, which occurs earlier in the pipeline and would be fixable downstream,

- **Representation washing** as defined by **?**, whereby low resource languages, while present in the corpus, are of a significantly lower quality than higher resource languages without any quality metric available publicly.

The most recent Common Crawl dump contains 64,000 shards. Each shard is composed of numerous records, and each record holds textual content along with metadata. While CommonCrawl shards hold document-level metadata that could be useful downstream, they were discarded and do not appear in OSCAR, whereas other corpora generated from the same source include them, e.g. CCNet (Wenzek et al., 2020). This limits OSCAR users to the textual content only, whereas metadata could have been distributed along with the corpus itself.

---

[25]OSCAR-orig: 6.3TB, OSCAR-dedup: 3.2TB

2.10.2 GOCLASSY

OSCAR was built using *goclassy*, a high-performance asynchronous pipeline written in Go (Ortiz Suárez et al., 2019). However, it suffers from several caveats that makes the re-generation and update of the corpus relatively complex in practice.

While goclassy's source code is easily readable thanks to the choice of an uncluttered language and a pragmatic approach, the lack of structure in both the source and the project itself makes goclassy difficult to extend and maintain.

The pipeline is not functional out-of-the-box, as the user has to provide the compressed shards from CommonCrawl, manually install fasttext (**?**Joulin et al., 2017) and create specific directories by themselves, since only partial instructions are given in the supplied README file.

goclassy also makes heavy use of I/O, as data is saved and loaded repeatedly between steps; as an example, the identification step stores language identification data and individual sentences in two files, before generating the final files (one per language). Despite these limitations, goclassy's performance is good due to Go's emphasis on easy and efficient parallelization and inherent speed. The pipeline uses clever handling of file descriptors, limiting I/O calls cost in some parts.

2.11 BUILDING A NEW OSCAR-LIKE CORPUS

We introduce *Ungoliant*, a new corpus generation pipeline that, like goclassy, creates a large-scale multilingual text corpus from a CommonCrawl dump. Contrarily to goclassy, Ungoliant is fully modular, better structured, and highly parametrizable; thereby allowing comparisons between several parallelization strategies. A specific effort was put in testing and documentation. Parts of Ungoliant are heavily inspired by goclassy, although it is implemented in Rust rather than in Go, which is sometimes faster.[26]

Additionally, we use Ungoliant to generate a new corpus from a recent Common Crawl dump. The new corpus includes metadata information while retaining backward compatibility with the OSCAR corpus.

2.11.1 UNGOLIANT

RATIONALE AND SCOPE

While Ungoliant is heavily inspired by goclassy, it provides a better set of tools to download, process, filter and aggregate textual and contextual data from CommonCrawl. These operations can be sequential, parallel or both, depending on contexts and performance requirements.

---

[26]https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-go.html

Platform	#shards	goclassy	Ungoliant	Approx. speedup
Desktop	1	30s	13s	×2.3
	10	3m6s	2m12s	×1.3
	25	9m10s	5m47s	×1.5
HPC	1	40s	6s	×6.6
	25	2m40s	1m6s	×2.4
	100	7m59s	4m14s	×1.8

Table 2.3: Comparison of approximate generation times depending on platform and number of shards.

We provide both batch and streaming processing, so that the whole pipeline could be run either online, with every step running on streams of data, or offline, with every step running on tangible files, or a mix of both, using already downloaded CommonCrawl dumps but streaming the rest of the process. Moreover, we embed numerous filtering and deduplication utilities directly inside Ungoliant, making these features available for pipeline composition and post-processing.

Ungoliant features a loosely defined pipeline interface, on which we re-implement goclassy's one, while improving performance by threading more aggressively and avoiding I/O where it is not necessary: While goclassy uses intermediate files for tags and sentences, we try to keep everything in memory in order to avoid losing time loading or writing files. The Rust language provides constructs that helps us build complex abstractions and pipelines while limiting proactive file I/O or computing, since nearly all the reimplemented pipeline is built around lazy evaluation. File I/O is only used when loading shards, and when writing sentences in language files.

Through benchmarking we found that the best parallelization strategy is to use rayon[27], a work-stealing (**?**) parallel and concurrent library enabling massive parallelization. We parallelize on shard-, record- and sentence-level processing.

To evaluate Ungoliant performance, we run both goclassy and Ungoliant's implementation on 1, 10, 25 and 100 Common Crawl shards both on a middle-range laptop computer (i5-7200u, 8GB RAM, NVMe SSD) and a HPC node (Xeon 5218 (64 Threads), 180GB RAM). Results are shown in Table 2.3.

Ungoliant performs better than goclassy on all tasks, independently of the platform or number of shards processed. However, we can note that Ungoliant's speedup is higher on short tasks, which is explained by its aggressive multithreading strategy, while goclassy uses a record-scope multithreading at its finest granularity.

2.11.2 Iterating on the goclassy pipeline

CommonCrawl dumps contain metadata that hold useful information such as related records, recognized language(s), or origin URLs. Since OSCAR pipeline dis-

---
[27]https://github.com/rayon-rs/rayon

cards metadata and sentences can be shuffled, we lose the ability to investigate those metadata themselves, as well as working on potentially multilingual documents, since we separate text from metadata.

The new pipeline (and the resulting new corpus schema) aims to establish a first link between textual data and metadata from CommonCrawl, while staying backward compatible with the existing OSCAR schema.

In other words, switching from the original OSCAR corpus and the newly generated one should be a drop-in operation.

METADATA EXTRACTION AND LINKING

Our choice of keeping the corpus backward compatible with the original OSCAR introduces changes in the way the corpus is generated, namely regarding metadata: a record's body is composed of sentences that aren't guaranteed to be of the same language. Since OSCAR merges sentences from multiple records into a single file, special attention has to be paid to the metadata dispatch too.

Approaches to tackle this problem range from (1) storing all metadata in a single location to (2) having language-specific metadata files that contain the metadata for each line in the language file.

Both (1) and (2) have their strengths and weaknesses, namely:

1. Having all metadata at the same place may facilitate wide queries about whole metadata, but at a cost of a very large size (which harms both accessibility and performance).

2. Getting the metadata for a given line is fast since line numbers are synchronized, but there is repeated information and a potentially important increase in size.

We choose a hybrid approach which keeps metadata local to each language, while trying to limit the information repetition by keeping an entry by group of chunks rather than by line, where a chunk is a series of contiguous sentences that share the same language from the same document.

An overview of the pipeline can be seen in Figure **??**, with a more precise view on record processing and metadata extraction in Figure 2.2.

Metadata are distributed via JSON-encoded files holding an ordered list of metadata entries, along with offsets (o) and paragraph lengths (l), enabling any user to get the content of a said metadata by querying for lines $(o, o + l]$ in the content file.

This approach still has drawbacks, in particular when looking for the corresponding metadata of a given sentence/paragraph, where one has to perform a search on the metadata file, or when working with multilingual documents. Another drawback is the resulting cost of potentially merging back numerous language parts: Since metadata query is offset-based, merging back metadata files implies updating those offsets.
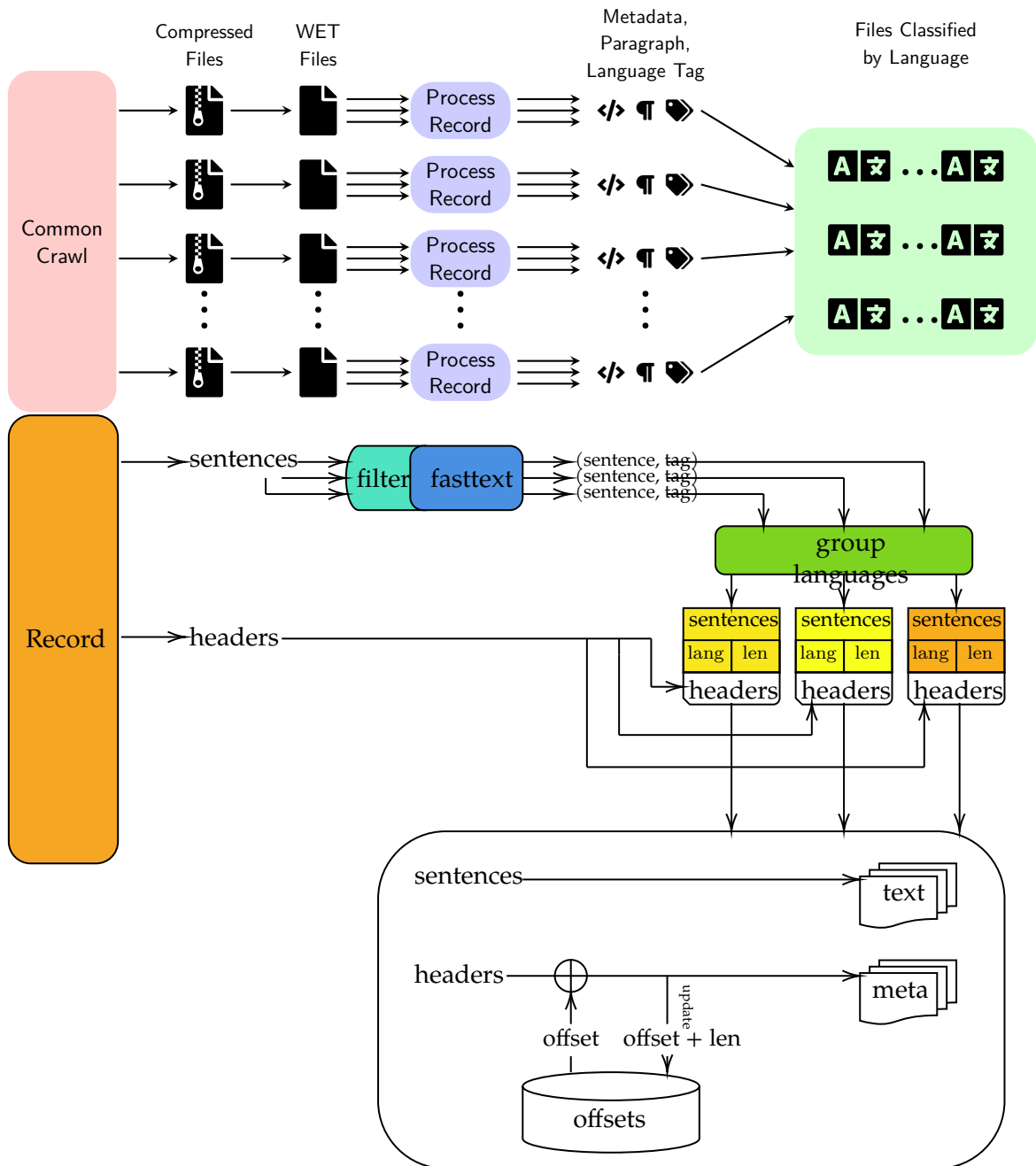
Figure 2.2: Record processing with metadata extraction. Headers are kept aside while sentences are identified and grouped into same-language bins. Headers are then cloned for each bin, and are sequentially stamped with an offset that is recorded for the whole operation, and written to disk into text and metadata files by language.

Platform	#shards	OSCAR	With Metadata	Speedup
Desktop	1	13s	12s	×1.1
	10	2m12s	1m55s	×1.1
	25	5m47s	4m50s	×1.2
HPC	1	6s	7s	×0.9
	25	1m6s	1m12s	×0.9
	100	4m14s	4m36s	×0.9

Table 2.4: Comparison of approximate generation times with and without metadata generation.

Having paragraphs and metadata linked by offsets in a highly parallelized pipeline implies to take special care at the offset level. The solution is to use shard-scoped offsets (starting from 0 for each language), and to keep global offsets protected by a mutex guard. This way, when a given shard is done processing and is ready to be written on disk, we convert shard-scoped offsets to global-scoped ones, update the global-scoped ones and then write text and metadata on disk.

We compare running times for the reimplementation of the goclassy pipeline, and our new pipeline adding metadata extraction, using both desktop and HPC contexts. The results are reported in Table 2.4.

Metadata generation does not seem to influence generation time dramatically. However, we can notice a slight performance difference between HPC and Desktop contexts. These differences may lie in the storage medium differences, I/O layout, or algorithmic peculiarities benefiting desktop contexts because of other bottlenecks.

2.11.3 Characteristics of our new backward compatible OSCAR-like corpus

We evaluate the newly generated corpus, assessing its ability to reflect events that occurred after the publication of OSCAR 2018 and detail the metadata format and potential use.

Comparison with OSCAR

While it is expected that our new corpus has a larger file size than OSCAR since CommonCrawl itself grew from 7.42TB to 8.06TB, metadata quickly adds up and take for nearly 15% of the whole uncompressed data.

The size augmentation is not the same for each language, and while the whole corpus is bigger now, some languages are smaller than they were before.

Results show that already largely represented languages gain more and more data (like the English language, which constitutes more than a third of the original OSCAR), except for the Russian language which loses approximately 100Gb of textual content. These results are summarized in Figure 2.3.

Version	Source	Textual (dedup)	Metadata	Total (increase)
2018	7.42TB	6.3TB (3.2TB)	N/A	6.3TB
2021	8.06TB	7.2TB (3.3TB)	1.2TB	8.4TB (+33%)

Table 2.5: Comparison of CommonCrawl and OSCAR sizes between 2018 and 2021 versions. Compressed (CommonCrawl) sources are from November 2018 and February 2021. Total is Textual + Metadata without deduplication.
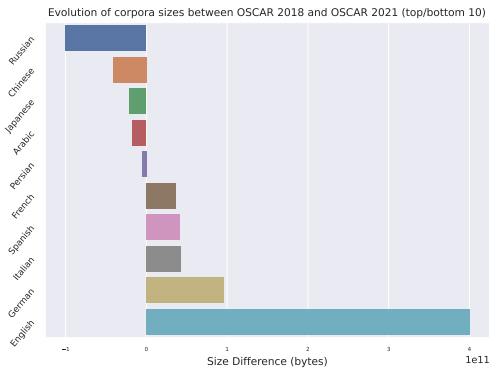

Figure 2.3: Comparison of language size (in bytes) between OSCAR 2018 and OSCAR 2021 (top/bottom 5 only).

However, in a context where the number of languages is very high (higher than 150) and of varying sizes, evolution can't be analyzed via a mere size evaluation. By computing, for each language, the relative size difference between the 2018 and 2021 releases of OSCAR, less resourced languages do appear, hinting at a better representation of some of them. These results can be found in Figure 2.4.

Numerous languages have been omitted from Figure 2.4, either:

- because they were present in the original OSCAR and are now absent (*Central Bikol* and *Cantonese*)

- because they were absent in the original OSCAR and are now present (*Manx*, *Rusyn*, *Scots* and *West Flemish*)

Precautions have to be taken when using these corpora and further work has to be done to correctly assess the quality of low-to-mid resource languages in order to better reflect the quality of each corpus to the OSCAR users. Some languages exhibited either a particularly low number of sentences or a very low quality, and as such couldn't be usable, while still accounting for a language in the total language count of the original OSCAR.
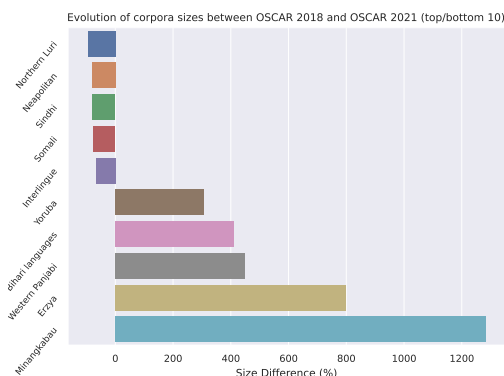
Figure 2.4: Comparison of language percentage between OSCAR 2018 and OSCAR 2021 (top/bottom 5 only).

Metadata

Metadata provides new contextual data that is useful to evaluate the corpus and draw metrics.

The total size of metadata is 1.2TB, ranging from 4Kb to 500Gb, depending on the number of lines. Relative size varies from 100% to 20%, diminishing with the textual data size, which is expected.

Metadata are provided in single files for now, but split versions of both textual and contextual data will be released soon after the release of the corpus, enabling easy access.

Our choice of keeping metadata aside from the main content adds some complexity when working with both textual and contextual data:

- When trying to get the metadata of given sentence, one has to get the line number k, then sequentially (or use a search algorithm since offsets are sorted) look for the record (with offset o and length l), where $k \in [o, o + l]$.

- Looking for lines corresponding to a particular metadata entry is easier: one has to read the textual file, skipping until the o-th line, then read l lines.

Presence of events

Using a sample of an English part of our corpus, we perform a simple search of terms in order to assess and compare the presence of pre- and post- 2018 events and persons in both corpora. Terms and frequency are grouped in Table 2.6.

Our corpus keeps around the same number of occurrences for pre-2018 events or public figures such as Barack Obama, while increasing the occurrence of people linked to more recent events (Joe Biden).

Language	Term	2018	2021
Arabic	Beirut port explosion	0	31
Burmese*	Min Aung Hlaing	387	3439
English	Obama	30039	27639
English	Biden	990	19299
French	Yellow Vests	2	96

Table 2.6: Comparison of occurrences of news-related terms between OSCAR and our corpus in a sample of 100 CommonCrawl shards. For the Burmese language, we use the whole 2018 and 2021 corpus since it is a low resource language. Terms are translated in the corpus language.

We include search terms linked to post-2018 events in French and Arabic which are smaller corpora (resp. 200 and 80 GB), and in Burmese, a mid-resource language (approximately 2GB). We observe a term occurrences evolution that reflects the linked events' timing and importance.

2.11.4 License

This new corpus will be released under a research-only license that is compliant with the EU's exceptions for research in text and data mining. Contrarily to the original OSCAR, no shuffled version of the corpus will be distributed, instead we will put in place an authentication system that will allow us to verify that requests for the corpus come from research institutions. A contact form will be also provided for independent researchers so that we can study their particular cases and determine if the utilization of the corpus corresponds to a legitimate research use.

Moreover, the introduction of metadata makes our corpus far more queryable, thus simplifying and speeding up the handling of take-down GDPR requests. For this reason, we will be releasing the complete set of metadata under a CC0 public domain license, so that any individual can check if their personal or even copyrighted data is in our new corpus and make a request accordingly.

2.12 Conclusion

We show that our solution is able to generate an OSCAR-like corpus that is augmented with metadata without breaking compatibility, while being faster, better tested and thoroughly documented. We believe our new pipeline and corpus will be useful for applications in computational linguistics as well as in corpus linguistics in general.

The generated corpus is of a larger size when including metadata and without deduplication. However, deduplicated textual content is of the same magnitude between OSCAR 2018 and OSCAR 2021, while reflecting topic changes from all over

the world. This fact suggests that old data may be lost with the time passing, and could be resolved by using CommonCrawl releases to build an incremental corpus, with every version augmenting the corpus size.

Metadata enables queries and statistics on the generated data, and we believe that it can be used to filter OSCAR to generate corpora that respond to certain criteria.

We plan to make this new version of OSCAR available under research constraints, with split versions of both textual content and metadata along with tools to operate on the corpus, enabling fast and easy operation on the corpus for researchers.

3   Modern French Data

3.1   LEM17

3.2   presto max

3.3   presto gold

4   Ancient/Medieval French Data

4.1 BERTrade Corpus

5 OTHER DATA

Part II

Models

6 CamemBERT

7 FrELMo

8 D'Alembert

9 BERTRADE

Part III

Downstream tasks

10 PARSING

11 POS Tagging

12 NAMED-ENTITY RECOGNITION

13   Text Normalization

14 DOCUMENT STRUCTURATION

Part IV

Real World Application

15   Le Petit Larousse

16 BASNUM

17 SCIENTIFIC PAPERS

18   Modern French Treebank

19 NAMED-ENTITY RECOGNITION CORPORA

Bibliography

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5360–5369, Hong Kong, China. Association for Computational Linguistics.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. 2020. ParaCrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Philipp Koehn, and Christof Monz. 2018. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels. Association for Computational Linguistics.

Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. 2009. Clueweb09 data set.

Isaac Caswell, Theresa Breiner, Daan van Esch, and Ankur Bapna. 2020. Language ID in the wild: Unexpected challenges on the path to a thousand-language web text corpus. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6588–6608, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-

length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

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

Ahmed El-Kishky, Vishrav Chaudhary, Francisco Guzmán, and Philipp Koehn. 2020. CCAligned: A massive collection of cross-lingual web-document pairs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5960–5969, Online. Association for Computational Linguistics.

Miquel Esplà, Mikel Forcada, Gema Ramírez-Sánchez, and Hieu Hoang. 2019. ParaCrawl: Web-scale parallel corpora for the languages of the EU. In *Proceedings of Machine Translation Summit XVII: Translator, Project and User Tracks*, pages 118–119, Dublin, Ireland. European Association for Machine Translation.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomás Mikolov. 2016. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Pedro Javier Ortiz Suárez, Laurent Romary, and Benoît Sagot. 2020. A monolingual approach to contextualized word embeddings for mid-resource languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1703–1714, Online. Association for Computational Linguistics.

Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. 2019. Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. In *Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019*, pages 9 – 16, Mannheim. Leibniz-Institut für Deutsche Sprache.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition, linguistic data consortium. *Technical report, Technical Report. Linguistic Data Consortium.*

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *OpenAI Blog*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1:8.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. 2021. WikiMatrix: Mining 135M parallel sentences in 1620 language pairs from Wikipedia. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1351–1361, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France. European Language Resources Association.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27.