CCpdf: Building a High Quality Corpus for Visually Rich Documents from Web Crawl Data

Michał Turski[1,2], Tomasz Stanisławek[1],
Karol Kaczmarek[1,2], Paweł Dyda[1,2], and Filip Graliński[1,2]

[1] Snowflake*
`firstname.lastname@snowflake.com`
`http://snowflake.com`
[2] Adam Mickiewicz University, Poznań, Poland
`firstname.lastname@amu.edu.pl`

Abstract. In recent years, the field of document understanding has progressed a lot. A significant part of this progress has been possible thanks to the use of language models pretrained on large amounts of documents. However, pretraining corpora used in the domain of document understanding are single domain, monolingual, or nonpublic. Our goal in this paper is to propose an efficient pipeline for creating a big-scale, diverse, multilingual corpus of PDF files from all over the Internet using Common Crawl, as PDF files are the most canonical types of documents as considered in document understanding. We analyzed extensively all of the steps of the pipeline and proposed a solution which is a trade-off between data quality and processing time. We also share a CCpdf corpus in a form or an index of PDF files along with a script for downloading them, which produces a collection useful for language model pretraining. The dataset and tools published with this paper offer researchers the opportunity to develop even better multilingual language models.

Keywords: Natural Language Processing, language models, dataset construction, document understanding.

1 Introduction

Natural Language Processing (NLP) in recent years has made significant progress thanks to using language models such as GPT-3 [6] or T5 [23]. Usually these models are trained in a two-step process. The first part is pretraining, which utilizes a large corpus of text, and the second step is finetuning on a final task. Recent works demonstrate a considerable impact of pretraining on the final performance of a model [10,17,27]. For instance, GPT-3 was pretrained on a combination of texts from Common Crawl, WebText2, two book corpora, and the English Wikipedia (499 billion tokens in total)[6] while T5 was pretrained on the C4 corpus, which is 750 GB of data[23].

* work done while at Applica.ai, later acquired by Snowflake

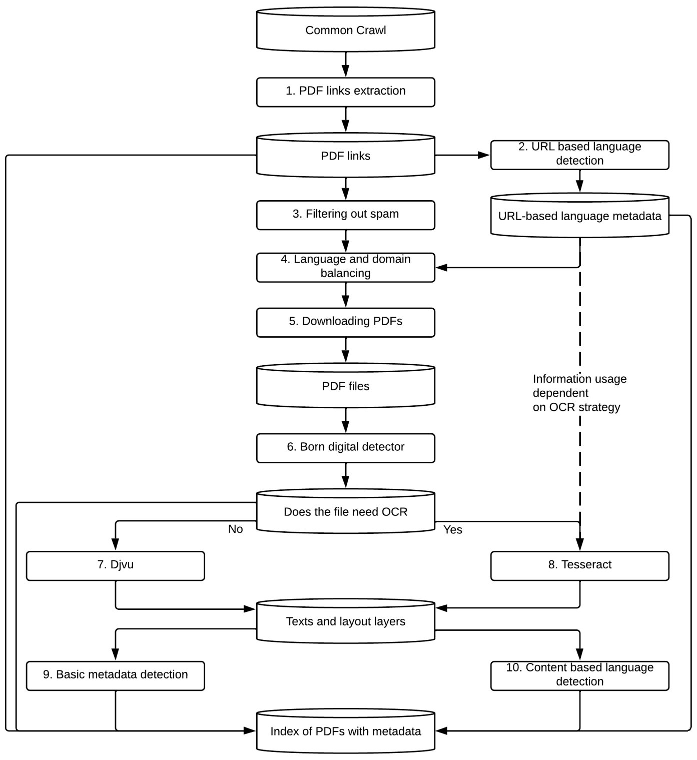
Common Crawl
1. PDF links extraction
PDF links
2. URL based language detection
URL-based language metadata
3. Filtering out spam
4. Language and domain balancing
5. Downloading PDFs
PDF files
Information usage dependent on OCR strategy
6. Born digital detector
Does the file need OCR
No
Yes
7. Djvu
8. Tesseract
Texts and layout layers
9. Basic metadata detection
10. Content based language detection
Index of PDFs with metadata

Fig. 1: The full flow of the process. Cylinders represent data, rectangles represent processing steps, and arrows represent data flow. A solid line indicates that the information is always used, and a dashed line represents data usage dependent on the processing strategy.

The recent progress in document understanding (defined as "capacity to convert a document into meaningful information" [5]) has been possible thanks to 2D language models such as LayoutLM [30,29], LAMBERT [9], or TILT [21]. Similarly to the models mentioned above, they also need large amounts of data for pretraining. The input to these models is a multi-modal representation of a document, e.g. tokens with their positions and images of pages.

The World Wide Web abounds in multi-modal documents, which contain enormous amounts of information. This information can be used in multiple domains: NLP, law, knowledge extraction, history, and many more. Yet, this aspect of the Internet remains relatively unexplored. So far, attempts of document dataset creation have been focused on either single domain (e.g. medical[3], academic [3], or industrial [14]), while there has been no all-over-the-Internet

[3] www.ncbi.nlm.nih.gov/pmc/tools/openftlist/

approach. On the other hand, existing all-over-the Internet corpora (e.g. Web 1T 5-gram[4]) were focused on text only, not on multi-modal documents.

A document is a multi-modal form of communication: to interpret documents properly, we have to understand not only text, but also the layout and graphical elements. The most popular and portable multi-modal document format is PDF. In this study, we aim to describe a carefully designed pipeline for PDF corpus creation. We investigated numerous possible processing techniques and described their impact on the final data, which allowed us to achieve satisfactory trade-off between data quality, computing time, and monetary cost. The dataset itself (in the form of an index of PDF files and a script for downloading them) is also available at our website[5]. We share a corpus of 14.5M pages. It is useful as a dataset for 2D language model pretraining, but may also be employed as a source for derived datasets, in the same way as the IIT-CDIP dataset [14] was used to create many diverse challenges. Finally, analysis of the collected PDF files themselves yields helpful insight for language model creators, but also enhances our understanding of the World Wide Web as a source of PDF documents.

2 Related works

The general problem of creating a large-scale corpus of documents has been studied extensively in recent years. IIT-CDIP [14] is a 40M pages (but according to the authors of OCR-IDL [4] only 35.5M of them are still reachable) dataset of reports from the Legacy Tobacco Documents Library[6] collection, which was later reused to prepare a 400k page document classification dataset [12]. Also, OCR-IDL [4] reused IIT-CDIP to publish a 26M page dataset with high-quality OCR output. DoRe [19] is a French dataset of 2350 annual reports from 336 companies, unfortunately the data weren't shared publicly. There are also two layout analysis datasets based on scientific articles: Docbank [15] and Publaynet [33]. Their volumes are 500k and 360k pages, respectively. In addition, Ammar *et al.* [3] provided corpora of scientific documents together with a literature graph (defined as "a directed property graph which summarizes key information in the literature and can be used to answer the queries mentioned earlier as well as more complex queries"[3]). The National Library of Medicine has shared a PMC Open Access Subset[7] which is a corpus of open-access, open-licensed medical publications. Allison *et al.* [2] proposed a pipeline for creating a corpus of PDFs sourced from the Internet. The goal of this work is to "identify key edge cases or common deviations from the format's specification". They also provide analyses of files in their corpus. All of these datasets are single-domain or single-language collections (usually both), while our aim is to create a diverse, multilingual dataset. There exists only one publication presenting such a dataset [31], but the authors limited

[4] `https://catalog.ldc.upenn.edu/LDC2006T13`
[5] `https://github.com/applicaai/CCpdf`
[6] `https://industrydocuments.ucsf.edu/tobacco/`
[7] `https://ncbi.nlm.nih.gov/pmc/tools/openftlist/`

themselves to describing the data processing pipeline without analyzing their decisions. Also, their dataset was not shared.

Attempts have also been undertaken to create diversified corpora of texts sourced from the Internet. For instance, in CCNet [28], Common Crawl was used to create curated monolingual corpora in more than 100 languages. Also Schwenk *et al.* used Common Crawl in CCMatrix [24], but their purpose was to extract parallel sentences in different languages. The result was 10.8 billion parallel sentences in 90 languages. Another study in this vein is Smith *et al.* [25], whose method allowed to extract a 278 million token corpus of parallel English-French, English-Spanish, and English-German texts. In CCQA [13], a method for composing multilingual question-answering task using Common Crawl was proposed. The authors shared 130 million question-answer pairs. Liu and Curran [16] used Open Directory Project[8] to extract a topic-diverse English corpus of 10 billion words. To pretrain the T5 language model [23], the authors extracted a 750 GB English text corpus, called C4, employing Common Crawl. Dodge *et al.* [7] explored this dataset further and analyzed the effects of the applied filtering. A similar pipeline to that used for C4 was applied to create the mT5 [32] training corpus, which is a multilingual version of T5. The proposed corpus has 6.3 trillion tokens. Qi *et al.* [22] crawled 10 million images with captions from the Internet and used it to pretrain the multi-modal ImageBERT model. C4Corpus [11] (not to be confused with C4 proposed by Raffel *et al.*, described above) utilized Common Crawl resources to provide multilingual (more than 50 languages) over 10 billion token corpus to the community. The Pile [8] is a 885 GB text corpus composed of 22 different datasets, and one of its subparts are texts from Common Crawl. Abadji *et al.* [1] proposed a document-oriented multilingual 12 GB corpus of texts from Common Crawl with quality annotations. It must be noted that the authors define the term "document" as a long, coherent piece of text, not as a PDF file, as we do in this study. Luccioni and Viviano [18] analyzed Common Crawl in terms of undesirable content, including hate speech and sexually explicit content, and investigated different filtering methods.

Dataset	**Documents**	**Pages**	**Avg pages per doc**	**Languages**	**Domains**	**Years**
IIT-CDIP	6.5M*	35.5M*	5.5	1	Industry documents	1990s
OCR-IDL	4.6M	26M	5.7	1	Industry documents	1990s
CCpdf	1.1M	14.5M	12.9	11	Multi-domain	Mostly 2010–2022

Table 1: Comparison of existing publicly available corpora. *Numbers of valid documents/pages according to the authors of OCR-IDL [4].

3 Collecting and processing PDFs

In this section we describe how we addressed the challenge of finding, downloading, and processing a great volume of PDF documents. The full process is presented in Figure 1.

[8] `http://odp.org`

3.1 Common Crawl

As our input we used web indexes created by Common Crawl[9]. Common Crawl is a project of The Internet Archive[10] – an organization dedicated to providing a copy of the Internet to the community. They crawl webpages and save them into crawls dumps. A crawl dump contains billions of webpages (hundreds of terabytes of uncompressed data) and a new dump has been published nearly every month since March 2014. Some earlier, more irregular dumps starting from 2008 are also available.[11] Each dump also contains an index of the crawled pages.

We decided to simply use the latest (and the largest) dump available at the time of writing this paper — the May 2022 dump.[12] It contains 3.45 billion web pages, which amounts to 462 TB of uncompressed content. It would obviously be possible to apply the extraction procedure described in this paper to all crawls to obtain an even larger collection of PDFs, which would also allow for a diachronic analysis, but we wanted to focus on the most recent documents.

Note that dumps contain only files considered as text files by the Common Crawl web robot. Mostly these are web pages in the HTML format, but, fortunately, PDFs are also treated as text files, being derivative of the PostScript page description language. This is not the case with, for instance, images, Excel files, DOCX files. Consequently, such files cannot be amassed using the methods described in the aforementioned papers.

3.2 PDF links extraction

We experimented with two methods for extracting links to PDF files (step 1 in Figure 1):

1. using CDX files, i.e., index server files provided by Common Crawl;
2. looking for links to PDF files in WARC, i.e., raw crawl data files.

The first method is simpler, as CDX files are easy to download and take up only 225 GB in total. The second method might yield more links to PDF files, but:

– it is impossible for us to download all WARCs. Only a limited number of them can be processed, though still a significant number of PDF links can be added even if a small percentage of all WARC files are processed,
– there is lower probability that the file linked is available at all, be it in the crawl dump or simply at the original address.

In CDX files, the MIME type of a captured file is specified, and we limited ourselves to the `application/pdf` type.

Hence, in this paper, we focus on the first method, which allows to speed up the whole processing pipeline.

[9] `https://commoncrawl.org`
[10] `https://archive.org/`
[11] `https://commoncrawl.org/the-data/get-started/`
[12] `https://commoncrawl.org/2022/06/may-2022-crawl-archive-now-available/`

3.3 URL-based language detection

We decided to limit our investigation to the following set of 11 languages: Arabic, Dutch, English, French, German, Italian, Japanese, Polish, Portuguese, Russian, and Spanish.

When deciding whether to process a given URL, we applied a number of simple heuristics to determine the language. For example, we assumed that PDFs from `.pl` domains are Polish unless there is `lang=en` inside the URL etc. Note that this is a preliminary filter; later, when the contents have been downloaded, we do a proper language detection (see Section 3.9).

In August 2018, Common Crawl added language metadata to CDX files.[13] Unfortunately, the Compact Language Detector 2 employed there is applicable only for plain texts or HTMLs, and only a small percentage of PDF links contained the language metadata; therefore, it was unusable for our purposes.

This step of the pipeline is presented as block 2 in Figure 1.

3.4 Filtering out spam

One of the challenges to be tackled in Web information retrieval or when creating a massive text corpus sourced from the Web is the problem of (web) spam and, more generally, low quality pages (step 3 in Figure 1). Web spam is usually related to black-hat search engine optimization, i.e., creating link farms of web pages with automatically or semi-automatically generated content. It turns out that PDF files found on the Internet have the advantage of a relatively low percentage of spam, especially when compared to HTML web pages. More generally, we believe PDF files usually contain more formal content as most of them are business, legal, or scientific documents.

Still, some spam PDFs were found in Common Crawl dumps. Fortunately, the way in which spammers operate is rather homogeneous. A typical telltale of a spammy PDF was a long name composed of lower-case letters interspersed with hyphens. A regular expression was written to detect suspicious URLs, and if a domain happened to contain a large percentage of such URLs, it was assumed to be spammy as a whole and totally discarded. Thanks to this simple heuristic, in a sample of 1k documents we manually annotated (see Section 3.9) we found no spam PDFs.

3.5 PDF data download methods

In order to ensure diversity, we downloaded at most three PDF files from each domain for a language in a random but reproducible manner. For English and German this number was lowered to, respectively, one and two, as PDFs in these two languages are much more numerous compared to others. This limitation also serves as a filter against anomalies such as millions of PDFs coming from a single domain; especially a spammy one, if not detected with the procedure described in Section 3.4. Balancing is represented as step 4 in Figure 1.

[13] `https://commoncrawl.org/2018/08/august-2018-crawl-archive-now-available/`

The files were downloaded from the original URLs (step 5 in Figure 1). Optionally, one could extract the file from a Common Crawl dump, especially if the file is not available at the original site. We provide a script to extract PDF files directly from the dump; fortunately, one does not need to download the whole dump to extract a file.

There is, however, one serious issue with extracting PDFs from crawl dumps: all files are truncated by the crawler to 1 MB. This limit is quite high for HTML pages, but unfortunately rather low for PDF files. This means that only small-sized PDF files can be extracted from Common Crawl dumps; larger ones have to be downloaded from the original sites.

The final and intermediary statistics for the files downloaded are presented in Table 2.

	URLs found	**Anti-spam filtered**	**Domain balanced**	**Language balanced**	**Successfully downloaded**	**Successfully processed**
ar	65 395	65 374	13 142	13 142	11 710 (89.10%)	10 826 (82.38%)
de	1 661 317	1 659 713	320 978	200 000	182 607 (91.30%)	172 668 (86.33%)
en	11 515 766	11 501 781	952 776	200 000	182 071 (91.04%)	175 440 (87.72%)
es	871 843	871 478	106 143	106 143	93 163 (87.77%)	88 952 (83.80%)
fr	654 250	653 120	143 020	143 020	129 927 (90.85%)	121 905 (85.24%)
it	831 344	831 026	129 610	129 610	119 731 (92.38%)	114 265 (88.16%)
ja	1 160 543	1 160 410	151 686	151 686	139 990 (92.29%)	134 310 (88.54%)
nl	339 519	338 946	92 372	92 372	84 848 (91.85%)	79 720 (86.30%)
pl	438 770	438 531	85 635	85 635	79 668 (93.03%)	75 374 (88.02%)
pt	697 535	697 285	73 130	73 130	64 725 (88.51%)	61 405 (83.97%)
ru	628 473	628 061	105 535	105 535	91 708 (86.90%)	85 552 (81.07%)
all	18 864 755	18 845 725	2 174 027	1 300 273	1 180 148 (90.76%)	1 120 417 (86.17%)

Table 2: Number of documents per processing step and language. Percentage values show success rates of downloading (in the downloaded column) or downloading and processing together (in the processed column). The success rate for processing a downloaded document equals 94.94%.

3.6 Born Digital scanner

To process correctly all kinds of documents in the document understanding domain we need to extract tokens from PDF files with their bounding boxes sorted properly, i.e., according to the reading order. The most common approach [30,29,9,21] is to process each PDF file with the use of some OCR engine, e.g. Tesseract [26], Amazon Textract[14], Microsoft Azure Computer Vision API,[15] or Google Vision API[16]. This method simplifies the processing pipeline and removes the need to understand the complicated PDF file format.

The biggest challenge in direct text and layout extraction lies in processing image content since there is no easy way to detect whether an image contains text. On the other hand, some documents lack pictures altogether; instead they contain textual information in the PDF file structure. We call them *documents that do not require OCR*. From such documents text can be extracted along

[14] `https://aws.amazon.com/textract/`

[15] `https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview-ocr`

[16] `https://cloud.google.com/vision/docs/pdf`

with bounding boxes using dedicated Python libraries, such as pdfminer.six[17], pdfplumber,[18] or a DjVu-based tool[19]. Direct text extraction using these tools leads to the reduction of the processing time and improvement of the quality of the extracted data by preventing OCR errors. Therefore, we decided to introduce a mechanism, called the Born Digital detector, for finding these kinds of documents (step 6 in Figure 1).

3.7 Born Digital detection heuristics

In order to detect documents that do not need to be processed with an OCR pipeline, we created a fast, simple heuristic-based classifier:

- *Visible Text Length* > 100 – Visible text in the document contains more than 100 characters
- *Hidden Text Length* $= 0$ – There is no hidden text in the document
- *Image Count* $= 0$ – There are no images in the document

Used statistics (*Visible Text Length*, *Hidden Text Length*, *Image Count*) were extracted using *Digital-born PDF Scanner* [20] tool written by us.

Our simple method was able to classify 219 documents out of 967 as born-digital files that do not require OCR. (In other words, we can skip the time-consuming OCR process for more than 1 out of 5 PDF files). To check quality of our heuristic we manually annotated the same sample of documents. The precision of the proposed method was 93.15%. All errors (15) were caused by adding a background with logo text to the file. In the future, we can also improve that kind of cases by extracting metadata information about PDF file background as well.

	Gold standard #	**Born Digital detector**			
		Precision	**Recall**	**F1-score**	**TP + FP #**
born digital, OCR not required	471 (48.71%)	93.15	43.31	59.13	219 (22.65%)
born digital, OCR required	321 (33.20%)	-	-	-	-
scan	175 (18.10%)	-	-	-	-
all	967	-	-	-	-

Table 3: Results for the Born Digital detector mechanism.

3.8 OCR processing

One of the initial steps of the PDF processing pipeline is the URL based language detection method (see Section 3.3). Information about the language of the document is needed for filtering documents for specific languages and also

[17] `https://github.com/pdfminer/pdfminer.six`
[18] `https://github.com/jsvine/pdfplumber`
[19] `http://jwilk.net/software/pdf2djvu`, `https://github.com/jwilk/ocrodjvu`
[20] `https://github.com/applicaai/digital-born-pdf-scanner`

by the OCR tool. In the next step (see Section 3.7), we select PDF files for processing either by the DjVu-based tool (if the file is born digital then it does not require OCR) or by Tesseract OCR [26]. The result is hocr files containing extracted text with its bounding boxes. This form of data serves as the input to the subsequent processing and analyzing steps.

Strategy name	**Processing time (using 1 CPU)**		**Additional cost**	
	1k files	in relation to fastest	Single page	1k files
DjVu-based tool + Born-digital detector	5.6 h	1x	-	-
Tesseract + URL based LD	23.7 h	4x	-	-
Tesseract + Built-in LD mechanism	75.9 h	14x	-	-
MSOCR + Built-in LD mechanism	16.7 h*	3x	$0.001	$13

Table 4: Comparison of resource utilization for different strategies of the text extraction from PDF files. *for Azure OCR we used 4 CPU (which is minimal recommendation for container in version 3.2) and multiplied the number by 4.

Possible alternatives In a typical scenario of extracting text with bounding boxes from a PDF file, researchers use a custom OCR engine [30,29,9,21], e.g. Tesseract, Microsoft Azure Computer Vision API, Amazon Textract, or Google Vision API. However, when we want to process millions of PDF files, we need to think about the utilization of resources in the context of time and money. Additionally, contrary to previous work, the language of a PDF file that we want to process is unknown. Therefore, to choose the most economical option, we tested the following strategies:

1. *DjVu-based tool with a born-digital detector* – for details, please see Section 3.7
2. *Tesseract with URL based Language Detection (LD)* – described at the beginning of this section
3. *Tesseract with a built-in LD mechanism* – in this strategy, we use the Tesseract OCR [26] engine with a built-in language detection mechanism
4. *Azure CV API with a built-in LD mechanism* – in this strategy we use Microsoft Azure Computer Vision API[21] with a built-in language detection mechanism

We achieved the shortest processing time (see Table 4) with the DjVu-based tool and a born-digital detector (see Section 3.7), which followed from the fact that we did not need to run any ML models. Also quality of output from the DjVu-based tool is better than from any OCR engine, because it extracts real content of a file and does not introduce any processing noise. *Azure CV API* and *Tesseract with URL based language detection* are the slowest OCR engines with 3-4 longer processing time. It turns out that the slowest processing strategy is *Tesseract with built-in LD mechanism*, and therefore, we will not apply it in our final pipeline.

[21] `https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview-ocr`

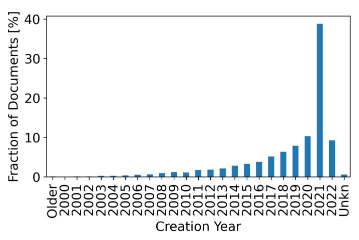

Fig. 2: Distribution of the analyzed sample in terms of creation year.

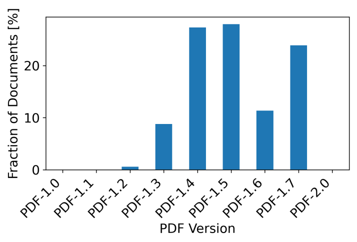

Fig. 3: Distribution of the analyzed sample according to PDF version.

3.9 Language Identification

In our final processing pipeline we used two language detection mechanisms:

1. URL-based method. Described in Section 3.3.
2. Content based method. We used the *langdetect* [22] library to detect language based on its text content extracted in the previous step.

We tested the quality of our language detection methods on ~1k manually annotated documents (Table 5). Both of our mechanisms can detect only a single language but, in reality, we found out that 27 documents had multiple languages (in 23 cases one of them was English). Fortunately, detecting a single language allowed us to predict the language correctly for almost all documents (97.3%).

With the use of the URL based method, we achieved a 90.51% F1-score on average, which seems reasonably good when we take into account the simplicity of the method. The content based method works better in general with an F1-score of 94.21% on average. The single exception here is the Japanese language. Both mechanisms produced the least satisfactory results for two languages: Arabic and English. It turned out that many documents from the *.ar* domain were actually in English. Therefore, for the content based mechanism we wrongly processed the PDF files with the Arabic Tesseract model.

Additionally, we found out that when we used the proper Tesseract model our results increased drastically to an F1-score of 98.05% on average. The main reason why this happened was the fact that the language identification mechanism was working on the proper alphabet.

Possible alternatives In Table 6 we present the results for different language identification tools. All of them achieved similar F1-scores, of which *spacy* (94.33%) and *langdetect* (94.21%) performed best. When we also take into consideration the processing time, it turns out that *gcld3* was the best one with a huge advantage over the second tool, which was the *langdetect* library. Therefore, we decided to balance quality and resource utilization and use *langdetect* as our main tool for language identification.

[22] `https://pypi.org/project/langdetect/`

[23] `https://pypi.org/project/langdetect/`

	Gold standard #	**URL based method**			**Content based method**					
					All documents			**Proper Tesseract lang**		
		Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1-score**
ar	20	46.51	95.24	62.50	44.19	95.00	60.32	100.0	100.0	100.0
de	94	94.68	92.71	93.68	98.94	98.94	98.94	100.0	100.0	100.0
en	119	80.46	58.82	67.96	94.34	84.03	88.89	98.55	98.55	98.55
es	75	94.52	93.24	93.88	98.65	97.33	97.99	98.57	98.57	98.57
fr	108	93.94	86.92	90.29	100.0	91.67	95.65	100.0	100.0	100.0
it	101	93.20	95.05	94.12	98.97	95.05	96.97	94.79	94.79	94.79
jp	108	100.0	98.10	99.04	100.0	89.81	94.63	92.38	92.38	92.38
nl	90	84.91	100.0	91.84	98.86	96.67	97.75	96.67	96.67	96.67
pl	88	95.56	100.0	97.73	98.86	98.86	98.86	98.86	98.86	98.86
pt	83	94.38	98.82	96.55	97.62	98.80	98.21	98.78	98.78	98.78
ru	78	96.34	98.75	97.53	97.47	98.72	98.09	100.0	100.0	100.0
other	2	0	0	0	0	0	0	0	0	0
no text	3	0	0	0	0	0	0	0	0	0
multi	27	0	0	0	0	0	0	0	0	0
all	996	88.59	92.51	90.51	93.45	94.99	94.21	98.05	98.05	98.05

Table 5: Quality of the language identification methods verified on 996 manually annotated documents.

Tool name	**F1-scores for content based method**												**Processing time**	
	ar	de	en	es	fr	it	jp	nl	pl	pt	ru	all	1k files	1M files
langdetect[23]	60.32	98.94	88.89	97.99	95.65	96.97	94.63	97.75	98.86	98.21	98.09	**94.21**	**0.28min**	**4.67h**
lingua-py[24]	60.32	97.90	88.79	96.69	94.74	95.92	98.59	96.77	99.44	98.18	97.47	94.05	2.57min	42.8h
spacy[25]	60.32	98.94	87.33	97.99	94.79	97.49	98.59	97.18	100.0	98.18	97.47	**94.33**	3.62min	60.3h
gcld3[26]	59.01	98.94	89.91	97.96	96.15	97.49	92.16	97.73	99.44	98.78	98.07	94.08	**0.03min**	**0.33h**

Table 6: Comparison of the quality and processing time of different language identification tools.

3.10 Produced index

As a result of our pipeline, we created an index of successfully downloaded and processed files. We decided to download up to 200k documents per language to share a reasonably sized corpus, with a good diversity of languages. It gives an acceptably good trade-off between the balance of languages and the size of the dataset. Statistics about the index are presented in Table 2.

A comparison of our dataset to existing corpora is presented in Table 1. The corpus we provided is smaller than the previous ones considering the total number of documents and pages. Still, language models will benefit in many aspects, (1) understanding long-distance relationships as the dataset has, on average, the longest documents compared to previous works, (2) multi-language training as we selected 11 different languages, (3) multi-domain training as we sourced documents from different websites all over the Internet, (4) document understanding of recently created documents (which may differ from the old ones in terms of language, layout, and graphical style) as the majority of files in our corpus were produced after 2010 (in IIT-CDIP, the most popular corpus so far, all the documents were created in the 90s).

[24] `https://github.com/pemistahl/lingua-py`

[25] `https://spacy.io/universe/project/spacy_fastlang`

[26] `https://pypi.org/project/gcld3/`

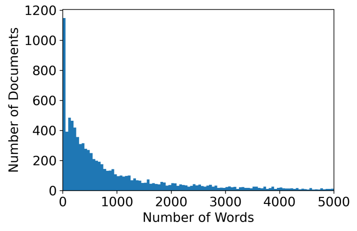

Fig. 4: Distribution of word count per document.

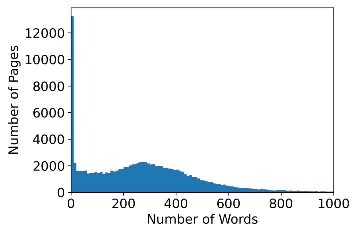

Fig. 5: Distribution of word count per page.

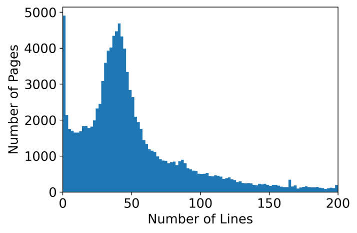

Fig. 6: Distribution of the number of lines per page.

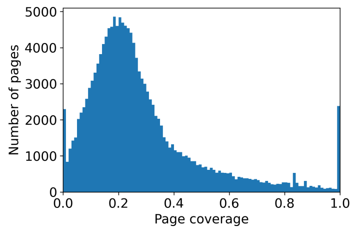

Fig. 7: Distribution of text coverage of page.

4 Exploration of PDFs

Since we provide a large scale, highly diversified collection of PDFs downloaded from all over the Internet, we want to provide some insight into the properties of PDF files which are accessible on the Internet. To do so we randomly picked 1k documents in each of the languages in our corpus (11k documents at total) and analyzed them in terms of various properties.

Firstly, we analyzed them in terms of their creation date, the outcome of which is presented in Figure 2. For this analysis we used the `CreationDate` field of metadata. Since most documents come with this field filled in, we were able to read the creation date for more than 99.4% of our sample. However, sometimes unreasonable values such as 1442 occurred as the creation year. As we can see, our corpus contains relatively new documents. It is an important point, because language evolves constantly, and three years ago terms such as "lockdown" or "post-pandemic" were absent from documents. Since we want our language models to represent current language and document types correctly, hence a distribution like that in Figure 2 is desired. The spike for the year 2021 was probably caused by the use of the Common Crawl dump from May 2022. We assume that crawlers from Common Crawl usually tend to find files that are a few months old, which often means that they are from the previous year.

We also analyzed the documents in terms of the exact version of PDF standard used. The data is presented in Figure 3. As we can see, the majority of our sample (above 76%) are PDFs prior to the 1.7 version. It is an important property, because versions 1.7 and 2.0 are defined by an ISO standard, while the older ones were defined only by Adobe proprietary standards. Some of the issues that we experienced during processing may have been caused by problems with older standards.

A PDF file contains metadata about the tool used to create it (Figure 8). There are many different tools, and often the same tool was described by different values (for instance Microsoft Word has different names in many languages despite being the same program). The two most popular providers of PDF tools found in our corpus are Microsoft (29.8% of the sample) and Adobe (21.3% of the sample).

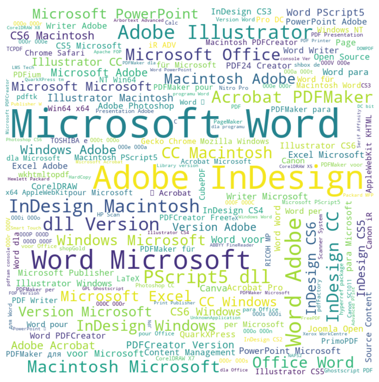

Fig. 8: Tools used to create PDFs.

Other properties that we were interested in were the length of the documents in terms of word count (Figure 4), their word count per page (Figure 5), and line count per page (Figure 6). As we can see, there is great variability in terms of these parameters. For instance, there are many documents and pages with almost no text. Up to our manual check, most of the documents with little text are graphically rich, for instance, technical drawings, or infographics. The typical value of words per page is between 0 and 500, and the typical value of lines per page is between 0 and 55.

To provide some insight into the layout of the documents, we checked to what extent each page was covered by bounding boxes of tokens. We may look at it as part of the text coverage parameter. Distribution of this value is presented in Figure 7. 76.2% of our sample fell into the range of 5% to 40% with respect to that parameter. Similarly to the previously described properties, once again we see a peak for empty pages. There is also a peak for pages fully or almost fully covered by text.

We were also interested in the ratio of page dimensions. 99.7% of x/y ratios were in the range of 0.4 to 2; the smallest value being 0.09, and the largest – 4.79. In our sample, 65.0% were pages with the dimension ratio close to $\sqrt{2}$, which is a standard ratio for the A, B and C paper series. 86.9% of them were vertical pages, and 13.1% – horizontal ones. Also, the LETTER format was popular; it comprised 10.6% of the sample: 92.9% documents were vertical, and 7.1% horizontal. In total, the A, B, C, and LETTER series comprised 75.5% of the sample.

To gather more information about the layout of the documents, we created heatmaps of token bounding boxes for vertical and horizontal pages (Figures 9 and 10, respectively). As we can see, layouts with two columns of text are fairly popular, especially for horizontal pages. Also, text occurs more frequently on the left side of a page.

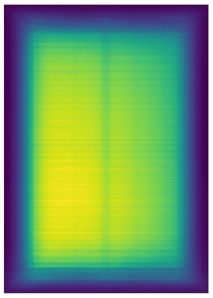
Fig. 9: Heatmap for vertical pages (brighter means more tokens, darker – fewer tokens).

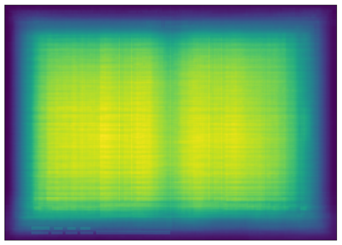
Fig. 10: Heatmap for horizontal pages (brighter means more tokens, darker – fewer tokens).

5 Discussion

In this study we analyzed a pipeline for creating a corpus of documents. According to our experiments, the most effective way of OCR processing of PDF files is a two-step procedure. The first step consists in the classification of the files according to whether they need an OCR engine or simple text extraction is sufficient. In the second step, we process the file with either an OCR engine (in our case Tesseract) or an extraction tool (in our case the DjVu-based tool). In the former scenario, we also discovered that predefining the OCR language speed up the process substantially; unfortunately, this comes at some cost in terms of data quality. However, this cost may be mitigated by a simple heuristic which filters out documents where the predefined OCR language did not match the one discovered by the language detector. We also analyzed different language detection tools in terms of output quality and processing time, and discovered that the *langdetect* tool offered the best trade-off between these values.

One of the limitations of this research study was that we focused only on the processing pipeline without analyzing the impact of each project decision on the final language model. However, this kind of study would be very expensive, as it would require multiple pretrainings of a language model. Language model pretraining is a costly process in terms of money, time, and environmental impact [20].

Also, conclusions drawn from the analysis of our sample can hardly be generalized to the whole content of the Internet and only provide some insight, rather undisputed knowledge. This follows from the filtering procedure: we decided to down-sample document-rich domains and languages, therefore, statistics calculated on the whole content of the Internet may differ from the ones presented in this work.

The approach which we used to create the dataset may be reused to all of the previous Common Crawl dumps in the WARC format, of which there are 84 in