applied sciences

Article

Sentence Boundary Extraction from Scientific Literature of Electric Double Layer Capacitor Domain: Tools and Techniques

Md. Saef Ullah Miah [1,*], Junaida Sulaiman [1,2], Talha Bin Sarwar [1], Ateeqa Naseer [3], Fasiha Ashraf [4], Kamal Zuhairi Zamli [1] and Rajan Jose [5,6]

[1] Faculty of Computing, College of Computing and Applied Sciences, Universiti Malaysia Pahang, Pekan 26600, Malaysia; junaida@ump.edu.my (J.S.); talhasarwar40@gmail.com (T.B.S.); kamalz@ump.edu.my (K.Z.Z.)
[2] Center for Data Science and Artificial Intelligence (Data Science Center), Universiti Malaysia Pahang, Pekan 26600, Malaysia
[3] Department of Software Engineering, School of Systems and Technology, University of Management and Technology, Lahore 54782, Pakistan; ateeqa.naseer@umt.edu.pk
[4] Department of Computer Science, School of Systems and Technology, University of Management and Technology, Lahore 54782, Pakistan; fasiha.ashraf@umt.edu.pk
[5] Faculty of Industrial Sciences & Technology, Universiti Malaysia Pahang, Gambang 26300, Malaysia; rjose@ump.edu.my
[6] Center of Advanced Intelligent Materials, Universiti Malaysia Pahang, Kuantan 26300, Malaysia
* Correspondence: md.saefullah@gmail.com; Tel.: +60-105328096

Abstract: Given the growth of scientific literature on the web, particularly material science, acquiring data precisely from the literature has become more significant. Material information systems, or chemical information systems, play an essential role in discovering data, materials, or synthesis processes using the existing scientific literature. Processing and understanding the natural language of scientific literature is the backbone of these systems, which depend heavily on appropriate textual content. Appropriate textual content means a complete, meaningful sentence from a large chunk of textual content. The process of detecting the beginning and end of a sentence and extracting them as correct sentences is called sentence boundary extraction. The accurate extraction of sentence boundaries from PDF documents is essential for readability and natural language processing. Therefore, this study provides a comparative analysis of different tools for extracting PDF documents into text, which are available as Python libraries or packages and are widely used by the research community. The main objective is to find the most suitable technique among the available techniques that can correctly extract sentences from PDF files as text. The performance of the used techniques Pypdf2, Pdfminer.six, Pymupdf, Pdftotext, Tika, and Grobid is presented in terms of precision, recall, f-1 score, run time, and memory consumption. NLTK, Spacy, and Gensim Natural Language Processing (NLP) tools are used to identify sentence boundaries. Of all the techniques studied, the Grobid PDF extraction package using the NLP tool Spacy achieved the highest f-1 score of 93% and consumed the least amount of memory at 46.13 MegaBytes.

Keywords: sentence boundary extraction; PDF to text conversion; NLP in material science; material information system; material informatics; Materials 4.0; gensim; NLTK; Spacy

check for updates

Citation: Miah, M.S.U.; Sulaiman, J.; Sarwar, T.B.; Naseer, A.; Ashraf, F.; Zamli, K.Z.; Jose, R. Sentence Boundary Extraction from Scientific Literature of Electric Double Layer Capacitor Domain: Tools and Techniques. *Appl. Sci.* **2022**, *12*, 1352. https://doi.org/10.3390/app12031352

Academic Editors: Evgeny Nikulchev and Vladimir Borisovich Barakhnin

Received: 10 December 2021
Accepted: 24 January 2022
Published: 27 January 2022

1. Introduction

Natural Language Understanding (NLU) is a subfield of natural language processing (NLP) that focuses on more specific NLP tasks, such as semantic parsing, relation extraction, sentiment analysis, dialogue agents, paraphrasing, natural language interfaces, question answering, and summarization [1]. Recently, material informatics has been applied in various material science research areas. Among these researches are automatic material discovery from scientific documents, automatic discovery of synthesis processes from scientific documents, and extraction of material properties and values from scientific documents [2]. All these researches are based on NLP and NLU. The discovery of materials and synthesis processes belongs to the Named Entity Recognition tasks, and the discovery of relationships between properties and values belongs to the NLU tasks. In a broader sense, all these tasks belong to the NLP domain, which requires clean and understandable textual content [3].

Both NLU and NLP techniques are widely used in materials informatics with the expansion of Materials 4.0 [4] implementation. Kim et al. [5] extracted synthesis parameters of metal oxides from 12,000 scientific articles using ChemDataExtractor [6] and SpaCy [7] tools. Both tools are natural language processing frameworks designed for materials science and general NLP processing tasks, respectively. In addition, both tools are designed for use with the Python programming language, which is popular among NLP, NLU, and materials science researchers [8]. In another study by Kononova et al. [9] on recipes for the synthesis of inorganic materials from the textual content, ChemDataExtractor and Spacy are also used. There is further work by Friedrich et al. [10], Hiszpanski et al. [11] and Kuniyoshi et al. [12] that use NLP-based approaches to identify materials, synthesis processes, and associated structures of materials and synthesis parameters. These works in the field of materials science indicate that there is a growing need for NLP- and NLU-based tools and techniques in the field of materials science. Besides materials science, other systems such as the recommendation system for scientific articles [13] and the recommendation system for relevant documents [2] also benefit from NLP- and NLU-based tools. To make the best use of these tools, a lot of textual data is required on which the performance and understanding of these tools depend. Therefore, the noise-free textual content is very important for building such a material information system. In order to generate noise and clutter-free textual content from scientific documents, text processing is required. The first step of text processing starts with converting documents of different formats, such as PDF documents or Microsoft Word documents, into plain text. When building a material information system, text processing is a very important task for the entire pipeline of any material information system. A material information system can be any system that can process, store, and visualize material related information. For example, a material information system can provide lists of materials and synthesis processes used to produce a particular product from various scientific documents from different years. To create such systems, materials scientists and researchers must invest time and effort to find appropriate tools and techniques. This study will help these researchers to reduce their burden in finding suitable tools and techniques, starting with the extraction of sentence boundaries.

In natural language processing (NLP), working with text is one of the most important tasks. In most cases, work related to text processing depends on correctly identified sentences and words or tokens [14–17]. Text content exists in Portable Document Format (PDF), as do web pages, plain text files, Microsoft Office Word format files, character streams, Postscript files, and in many other forms [18]. Correctly recognizing complete sentences or sentence boundaries from any text format can speed up workflow and reduce the amount of text preprocessing tasks [19].

Correct and complete sentences are one of the prerequisites for both semantic and syntactic analysis of any textual content [20]. As mentioned earlier, a significant part of the text is found in PDF format, especially most of the scientific literature from different journals of different publishers. PDF files differ from traditional plain text files, in which text is structured in a semantic way like words or paragraphs. Instead, it is a layout-based format based on the positions of characters [21]. Therefore, it is important to know which tool is more efficient in extracting text in semantic format from PDF files, more specifically in extracting text in the form of a complete sentence, so that researchers do not have to invest more time to convert the extracted text into the semantic format.

Nowadays, NLP researchers need to extract text from PDF files using various programming languages and frameworks. There are many web-based services and PDF-to-text converters. However, they are not suitable for batch processing, such as converting thousands of files at once. This is one of the reasons why NLP researchers use programming languages other than web-based conversion services. The Python programming language is one of the preferred languages, as it is ranked first in the PYPL programming language rankings as of 14 November 2021. [8]. Therefore, in this study, different techniques for extracting PDF in text using Python core packages and Python wrapper packages with external services are considered to compare the correct detection of complete sentences/sentence boundaries. The selection of Python PDF libraries or packages is based on the number of downloads in the last 30 days from the top PyPI packages, updated on 1 November 2021 [22,23]. As far as we know, there is no work to date that compares the performance of different techniques for extracting text from PDF files to recognize complete sentences correctly. The three main contributions of this work are listed below:

- Recommend a better Python package for extracting text from PDF files;
- Recommend a better NLP framework for sentence boundary detection;
- Determine the most effective technique by comparing the performance of the techniques used in terms of runtime and memory consumption.

The remainder of the paper is organized as follows: The Background Study section discusses some related studies, the Methods section describes the methods used in this study, then the Results and Discussion are presented, and the Conclusion section concludes the paper.

2. Background Study

Various researches have been carried out for the extraction of PDF into text. One of them deals only with the extraction of metadata from PDF files. In [24], the authors propose a framework for extracting metadata from PDF files, developed using the Java language, which they claim is 9 to 10 times faster than other available tools. In this work, the authors focused on metadata extraction. However, they did not measure how much text is extracted as a complete sentence. According to the study of [25], there are two classifications for information extraction: one is metadata extraction, and another is key-insights extraction. For the extraction of metadata and key insights, there are various computational approaches such as Deep Learning, Naïve Bayes classification, Support Vector Machine (SVM), Conditional Random Fields (CRF), Hidden Markov Models (HMM), and rule-based approaches [25]. However, this study also does not suggest any specific method or combination of methods or techniques to get text data from PDF documents with proper sentence boundary.

Zelun et al. [26], proposed a novel system called pdf2latex that uses deep learning methods to extract mathematical expressions and plain text into the latex format and achieves 81.1% accuracy. In [27], the authors conducted a comparison between four different commercially available optical character recognition software in converting Hansard Malaysia parliamentary reports. In this study, the author also focused on the correctly recognized characters on each page of the PDF document, and no sentence boundaries were considered. Bast et al. [21] proposed a rule-based PDF extraction tool called "Icicite", which can detect paragraph boundaries, word boundaries, reading order, and body-text paragraphs. However, the authors did not mention sentence boundaries in their work and compared 14 PDF extraction tools. The 14 tools mentioned in this study, including the proposed "Icicite" tool, work to detect paragraph boundaries, line breaks, and characters but not to detect proper sentences. In addition, this study presents a comparative evaluation of several tools, including Pdftotext, Pdfminer, and Grobid, which also appear in our study. Of these three tools, Grobid performed better, which is also evident from our experimental results. The study by Duretec et al. [28] presented the evaluation of Tika, DocToText, and Xpdf tools. Among these tools, Tika achieved 58% accuracy in extracting text from PDF documents, in orderly extraction, which is close to our experimental result. The official benchmark for the Pymudf tool is included in the official documentation [29] of this tool. Our experimental results also support the benchmark in terms of the tool's text extraction capabilities.

Lipinski et al. [30] experimented with extracting metadata from scientific PDF documents using various tools and techniques. In this experiment, the authors used seven different tools for extracting metadata from scientific PDF documents and provided the comparative experimental results in terms of accuracy. In their experiment, the Grobid [31] tool performed better than the tools used by pdftotext [32] tool to extract text from PDF documents in extracting metadata than the other six tools, with 92% accuracy in title extraction. Grobid also performed better than the other tools in extracting other metadata, including authors, authors' last names, abstract, and year.

Mayank et al. [33] proposed a tool called OCR++ that also extracts metadata from the scientific literature. The authors made a comparison between grobid and the tool ocr++. In many cases, including title, author's first name, last name, and section heading extraction grobid performed better than the proposed system in terms of f-1 score. However, this study does not provide comparative results or suggest a technique for extracting sentence boundaries from scientific PDF documents.

George Sanchez [34] conducted a study of sentence boundary detection in legal texts. The author compared three techniques, including Punkt Model with user-defined abbreviations, Conditional Random Fields (CRF), and Deep Learning Neural Network (DNN). For the Punkt model, the author adapted the NLTK punkt method and achieved an accuracy of 72% in recognizing sentence boundaries, which is significantly lower than the result of CRF and DNN, which achieve 93% and 89%, respectively. The sentence boundary detection experiment was conducted with plain-text corpora converted from json format and does not include PDF extraction for corpora generation. Among the existing NLP tools, only the NLTK tool is used in this study. The other tools are based on the machine learning approach used in this study.

Pree Thiengburanathum [35] compared two machine learning-based methods, namely CRF and BiLSTM-CRF, for sentence boundary detection in Thai sentences. In this study, a plain text corpus from Thai web forums consisting of online product review data was used. This study used plain text data rather than data extracted from a PDF file. In addition, this study did not use existing sentence boundary detection tools, but the author developed his own tool based on machine learning. In this study, the CRF-based approach provided better results than the BiLSTM-CRF approach.

Most studies found on PDF-to-text extraction focus on extracting metadata and text content from scientific literature in PDF format. The extraction is done while preserving the structure of the paper, e.g., extracting text content from the introduction section of a paper and labeling or identifying the extracted text as an introduction or just the metadata including author names, title, year, and abstract. However, very few studies mentioned techniques or tools for extracting sentence boundaries. The studies that do mention sentence boundary extraction use custom tools based on machine learning approaches and use plain text datasets from fields other than materials science, especially EDLC, which do not involve text extraction from PDF documents. In this scenario, two questions need to be answered:

- Is the extracted textual content in a complete sentence form?
- Is a single sentence treated as multiple sentences?

Therefore, answering these two questions provided the opportunity for this research to discover the appropriate tools and techniques for extracting sentence boundaries from scientific PDF documents, using a combination of existing PDF-to-text conversion tools and NLP frameworks, in an attempt to reduce the time required by researchers to find the appropriate combinations. Complete sentence recognition plays an important role in an automatic text processing system because a complete sentence contains both syntactic and semantic information. Different studies have used different tools and techniques for PDF extraction and sentence boundary detection in the creation of different chemical or material information systems, without providing a rationale for their choice of tools and the performance of the tools for PDF extraction and sentence boundary detection [36–38]. Therefore, a comparison between the existing PDF and text extraction tools is required to find a suitable one. Therefore, this study will be useful to EDLC researchers in their decisions about PDF extraction tools and NLP packages. As far as we know, there is no formal analysis to date that compares various existing PDF-to-text conversion tools in conjunction with sentence boundary detection tools. More precisely, there is no work that deals with sentence boundary extraction from scientific literature in the EDLC domain. Thus, the novelty of this study lies in the comparative analysis of different tools and techniques for PDF-to-text conversion and sentence boundary extraction for the EDLC domain.

3. Methods

Since different PDF extraction tools are utilized for the sentence boundary identification, this section is split into two subsections; namely, (i) overview of the employed techniques and (ii) experimental procedure.

3.1. Overview of the Employed Techniques

Since Python packages are used in this study to extract text from PDF documents, various techniques are employed using Python core packages and Python wrapper packages with external services. This section gives a brief overview of the techniques used to extract PDF-to-text.

3.1.1. Core Python Packages

This section gives an insight into four techniques used for text extraction from PDF documents, namely Pypdf2 [39], Pdfminer.six [40], Pymupdf [41], and Pdftotext [32].

Pypdf2

Pypdf2 is a Python toolkit or package for working with PDF files in the Python programming language. It evolved from the package Pypdf [42], which is no longer maintained by its original creator. Pydpf2 can extract data from PDF files and split, merge, or manipulate PDF files. This package must be used in the Python programming language, and there is no single executable script available for this package. This package converts all text from each PDF file as a page; the section selection option is not available in this package. The Pypdf2 tool uses StringIO instead of FileStreams and is thus more memory efficient. To extract text with this tool, the PDF file must be loaded as a binary object using the tool's pdfFileObj. Then this object is parsed using the PdfFileReader method. The text is extracted from this object using the extractText method. This extractText method checks whether the binary object is a TextStringObject or not. If it is a TextStringObject, the tool saves the object as text and returns the text to the user. Algorithm 1 describes the steps to extract the text using the Pypdf2 tool.

Algorithm 1 Text extraction using Pypdf2 tool

```
Input: pdf file
Output: text file
1: Get the pdf file as binary object $pdfFileObj$
2: Assign the $pdfFileObj$ to $pdfReader$ object
3: Get the page number count from the $pdfReader$ object
4: for each page do
5:    extract text using $extractText$
6:    append $extracted_text$ in memory
7: end for
8: return $extracted\_text$
```

Pdfminer.six

Pdfminer-six is a fork of the original Pdfminer [43] package. This version is maintained by the community, as the original Pdfminer package is no longer supported. The Pdfminer-six package is focused on text extraction and analysis. It extracts data directly from PDF files. This package can be used with the Python programming language as well as with a single executable script. In addition to direct text conversion from PDF files, this package also provides functions for extracting tables, interactive forms in Acroform, images, and tagged content. This tool extracts text from PDF files using a layout analysis algorithm (LA) [44]. The LA works in three phases. First, it groups characters into words and lines, then it arranges lines into boxes, and finally, it arranges the boxes hierarchically and returns the text content. To extract text from PDF documents using the pdfmniner.six tool, the target PDF file is loaded as a binary object, and this binary object is then parsed using the PDFParser method. After parsing the document using the PDFResourceManager, it is split into different page objects from which the texts are extracted as StringIO objects. Finally, the texts are extracted from the StringIO object and saved as a plain text file. Algorithm 2 presents the flow for extracting text using Pdfminer.six package.

Algorithm 2 Text extraction using Pdfminer.six tool

```
Input: pdf file
Output: text file
1: Define $output\_string$ as $StringIO$
2: Get the pdf file as binary object by $PDFParser$
3: Initialize $PDFResourceManager$
4: Define $device = TextConverter(rsrcmgr, output_string, laparams = LAParams())$
5: Generate $PDFPageInterpreter$ with device and resource manager
6: for each page do
7:     process page
8:     append extracted text to $output\_string$
9: end for
10: Get extracted text from $output\_string$ and store as a plain text file
11: return text file
```

Pymupdf

It is a Python wrapper for mupdf [45], a lightweight toolkit for rendering PDF, xps, and ebooks. It provides various features for rendering PDF files with high quality and high processing speed. This package must be used in the Python programming language and does not provide a single executable script. Pymupdf extracts text from PDF documents using the tool's TextPage class. A TextPage consists of blocks, such as paragraphs. A block contains lines and their characters or an image. A line contains spans, and spans contain characters with identical font properties for all adjacent characters. To extract text with the Pymupdf tool, the target PDF file is loaded with the document object. Then, PageCount is initialized to split the document into pages. After that, the getText method is used for each page found to extract the text content and save it in a plain text file. Algorithm 3 shows the workflow of text extraction using this tool.

Algorithm 3 Text extraction using the Pymupdf tool

```
Input: pdf file
Output: text file
1: Load the pdf file as document object
2: Initialize $PageCount$ from the document object
3: for each page do
4:     extract text using $getText$
5:     append $extracted\_text$ in memory
6: end for
7: return $extracted\_text$
```

Pdftotext

Pdftotext is a simple Python package for extracting PDF-to-text. Although it is a Python package, Poppler [46] must be installed on the system before installing it via pip in Python. Poppler is a free PDF rendering library compliant with ISO 32000-1, and a fork of Xpdf-3.0 [47], developed by Derek Noonburg. This package must be used inside Python programs and has no single executable script. The Pdftotext package uses poppler to convert PDF-to-text. After loading the target PDF file as a binary object file, this tool uses poppler's PDF method to convert the PDF file to a plain text file. The poppler library is written in C++ language, so it takes less time than other tools. This tool first determines the layout of the PDF file, then converts the file into different pages and decodes the content into UTF-8 format. After that, it returns the PDF file as a plain text document for all pages. Algorithm 4 presents the workflow for converting text using the Pdftotext tool.

Algorithm 4 Text extraction using Pdftotext tool

```
    Input: pdf file
    Output: text file
1: Load the pdf file as binary object
2: Initialize Poppler $PDF$ method and return $page$
3: Decode $page$ to UTF-8 format
4: Return text contents as plain text file
```

3.1.2. External Services with Python Wrapper

In addition to these Python packages, another popular PDF extraction tool from the Apache Software Foundation, Tika [48], and Grobid [31], are also used for comparison.

Tika

Developed by the Apache Software Foundation, Tika is a content discovery and analysis framework written in the Java programming language. Tika is used for parsing text and metadata from various file types, including PDF, Microsoft PowerPoint, and Excel format. It provides a server for text extraction and has command-line tools. Since it has a server feature, it can be used with any programming language. It also provides a Python package that can be easily integrated with Python. Moreover, it has command-line tools that can be used as an executable script to convert PDF-to-text without integrating any programming language. The Java Runtime must be installed in the system to use tika in Python. Before using tika to extract text from PDF files, the Python script utilizing the tika tool initializes the tika virtual machine and then loads the PDF file into the parser object of the tika tool. After loading the document, tika automatically determines the file's language and data types, known as Multipurpose Internet Mail Extensions (MIME). Then, based on the language and data type, the parser module extracts the text content and metadata of the document and returns it as a text stream. Then the text stream is saved as a plain text file. Algorithm 5 shows the steps for extracting text from PDF documents using the tika tool.

Algorithm 5 Text extraction using Tika tool

```
    Input: pdf file
    Output: text file
1: Initialize the tika Virtual Machine
2: Load the pdf file into the $parser$
3: Detect Language and Data Type
4: Extract text content and metdata content
5: Store text content in plain text file
```

Grobid

It is an open-source machine learning-based library for extracting and parsing PDF documents. This library was developed with a focus on the extraction of scientific publications. Grobid has a very flexible content parsing capability. It has functions for extracting headers, references, citation context, name and affiliation, full-text extraction, and patent and non-patent reference extraction. Grobid was developed in the Java programming language and provided a service structure so that it can be integrated with any programming language. It also has a batch processing mode with an application programming interface. Grobid first converts the PDF content into a structured XML/TEI encoded format; then, a custom parser can convert the required texts into a plain text format. To parse a document, Grobid employs a series of sequence labeling models. This modular method allows the training data, features, text representations, and models to be tailored to the document's various hierarchical structures. Individual models keep a modest number of labels, but the complete cascade produces very comprehensive end-result structures when used together. To convert PDF documents to text files, the Grobid server must first be initialized. Then, the PDF documents are loaded via the application programming interface (API), and a parameter is specified depending on the extraction model, e.g., full-text processing, metadata processing, or bibliography processing. In response to the request, a restructured tei-xml file is returned from the Grobid server. Subsequently, the tei-xml file is parsed with a specially developed XML parser based on the required text content, and the parsed text content is stored in a plain text file. Algorithm 6 shows the steps for extracting text from PDF documents using the Grobid tool.

Algorithm 6 Text extraction using Grobid tool

```
  Input: pdf file
  Output: text file
1: Initialize the Grobid server
2: Load the pdf file via API
3: Specify the parameter depending on the extraction model
4: Extract text as a structured XML/TEI format file
5: Parse the file as per the requirement with custom developed tei-xml parser
```

Table 1 provides a comparison of the features of all PDF-to-text extraction packages used in this study. Features include executable script, section-by-section extraction, API service, web service, and third-party dependency for all packages.

Table 1. Comparison of the various features provided by the PDF-to-text extraction packages used in this study.

Package Name	Executable Script	Section-Wise Extraction	API Service	Web Service	Third Party Dependency
Pypdf2	✗	✗	✗	✗	✗
Pymupdf	✗	✗	✗	✗	✗
Pdfminer.six	✓	✗	✗	✗	✗
Pdftotext	✗	✗	✗	✗	✓
Tika	✓	✗	✗	✓	✗
Grobid	✓	✓	✓	✓	✗

3.1.3. NLP Packages

In this study, three well-known Python NLP packages are used to identify sentence boundaries. They all have their advantages and limitations in identifying sentence boundaries. Therefore, three packages are used and compared to find the package with the best performance in sentence boundary identification.

NLTK

NLTK stands for natural language toolkit. It is an open-source, free, and community-driven framework designed to work with human language data. NLTK provides a very rich interface to over 50 corpora and lexical resources [49]. For sentence boundary detection, NLTK uses a sentence tokenizer that relies on PunkSentenceTokenizer. This PunkSentenceTokenizer has been trained on many speech corpora [50] to provide better sentence tokenization. This tokenizer separates a text into a list of sentences by building a model for abbreviated words, collocations, and words that begin sentences using an unsupervised method. It is built to learn unsupervised parameters (a list of abbreviations, for example) from a corpus that is similar to the target domain. In this paper, the pre-trained standard model for English is used to tokenize the sentences and to study the sentence boundaries. To study the sentence boundary with NLTK, each extracted text document is tokenized with this sentence tokenizer, and the sentences are stored. Then the sentences are matched with the ground truth sentences for each document.

Gensim

Gensim is a free, open-source Python library that processes plain text using unsupervised machine learning algorithms [51]. Gensim stands for Generate Similar, which was originally developed for topic modeling. Gensim's sentence tokenizer is based on unsupervised machine learning techniques [52], which is used to identify the sentence boundary. This tokenizer is implemented with the split_sentences method from the gensim.summerization.texttcleaner class. Gensim's sentence tokenizer is strict with punctuation marks. When it encounters a punctuation mark, it splits it and marks it as another sentence. In addition, the tokenizer counts each new line it finds as a new sentence. These properties have a significant impact on identifying sentence boundaries and are evident from the experimental results.

Spacy

Spacy is an industrial-strength natural language processing framework in Python. It uses pre-trained transformer models and word vector models such as BERT and Word2Vec. It supports more than 64 languages. It also supports custom models based on PyTorch and TensorFlow. Sentencizer is a pipeline component of Spacy [53] for sentence boundary detection. Sentencizer uses a rule-based detection model instead of the statistical dependency parser model. To detect sentence boundaries with this Sentencizer component, it must be added to spacy's default NLP pipeline with a custom punctuation marker configuration. In this study, we do not provide a custom configuration but use the default configuration of the Sentencizer component. To identify the sentence boundary, converted text files are processed with this component. Then, the sentences obtained by the Sentencizer component are stored and matched with the ground truth sentences for each document.

3.2. Experimental Procedure

To begin the experiment, 10 papers are selected from the field of electric double layer capacitors (EDLC). All 10 documents are manually examined to determine the exact number of sentences they contain. This number of sentences per document or paper is utilized as the basis for the evaluation. In addition to the count, all sentences are stored in a plain text file for all 10 documents. After selecting the documents, the Python PDF libraries discussed in Section 3.1 are used to extract text from the PDF documents. After text extraction, various sentence tokenizers from different NLP frameworks such as NLTK [54], Gensim [55], and Spacy [56] are used to determine the number of correctly extracted sentences. Then, each sentence identified by the NLP frameworks is compared with the manually extracted correct sentences to confirm the identification process. Figure 1 shows the overview of the applied methodology.

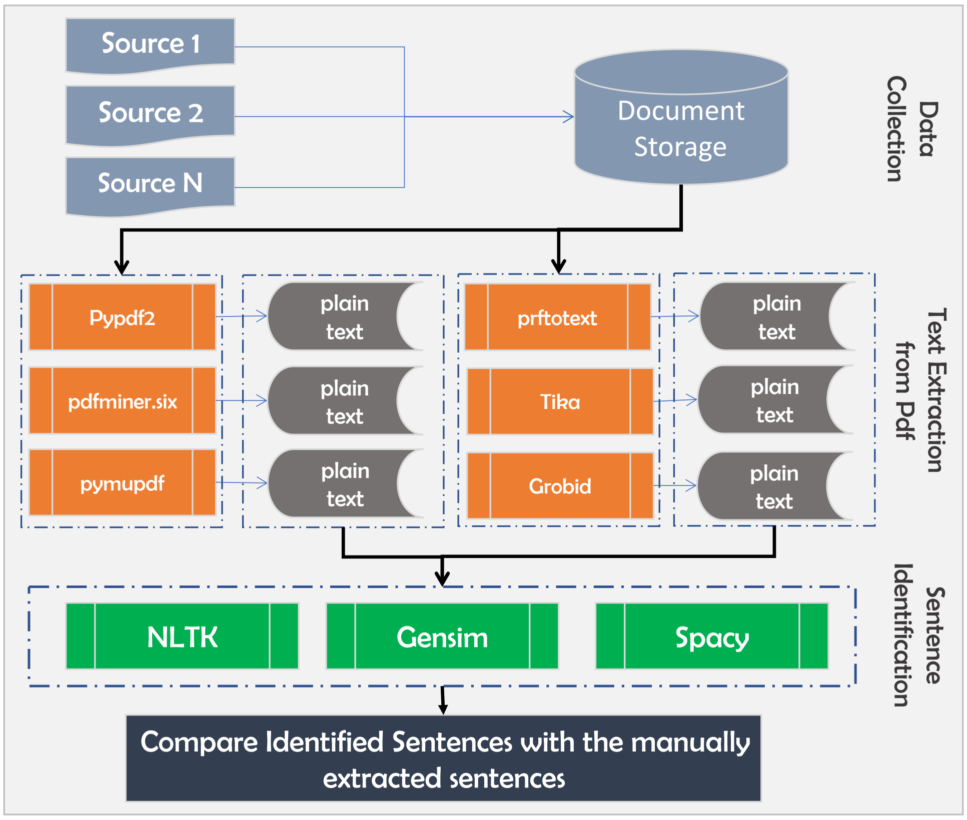

Figure 1. Overview of the methodology employed in this study to examine sentence boundary utilizing different PDF-to-text extraction Python tools.

3.2.1. Data Collection

For the experiment, the electric double layer capacitor (EDLC) field is considered. Therefore, a set of 10 scientific documents is collected in PDF format with different formatting styles and journal publishers, e.g., Journal of Power Sources, Carbon, and Electrochimica Acta. Then, each file is renamed chronologically from P1 to P10. After that, all the files are manually checked to find out the number of sentences in each file. In addition to recording the number of sentences, all sentences are stored in individual text files for all 10 documents. In this step, figures, tables, and references are excluded while collecting and counting the sentences. Table 2 contains the number of sentences for the collected documents.

Table 2. Document-wise sentence count for all the collected PDF documents.

Document	Sentence Count
P1	143
P2	119
P3	119
P4	350
P5	178
P6	81
P7	121
P8	153
P9	141
P10	86

3.2.2. Text Extraction from PDF Documents

Plain text is extracted from all collected PDF documents using the PDF-to-text extraction packages. All six Python packages are used in a single experimental setup developed as a Python program. In the program, all packages are defined as methods and are used to extract text from all documents. After extracting the text, they are saved as individual text files and stored in different folders for different packages. Then, these extracted text files are used to analyze the sentence boundaries and measure the performance of each package and tool.

3.2.3. Sentence Boundary Examination from Extracted Text

To analyze how many sentences can be recognized as complete sentences, three popular NLP packages for Python are used. The packages used are The Natural Language Toolkit (NLTK) [54], Gensim [55], and Spacy [56]. Each of these NLP frameworks has its own method for tokenizing sentences, which can be used to identify sentences from provided plaintexts. For this experiment, each package-specific sentence tokenization method is used to identify sentences from each converted text document. From the NLTK tokenize module, the sent_tokenize method, the sentencizer method from Spacy, and the split_sentences method from Gensim are used to test sentence boundaries. After applying the sentence tokenization methods from each of the NLP frameworks, all the sentences identified by the tokenizers are stored along with the total number of sentences identified. After obtaining the sentences identified by the tokenizers from each NLP framework, all sentences are matched against the ground truth sentences file for each paper using the Python difflib [57] package. For the sentence matching result, the matching ratio (R) is considered. If the R value is above 90%, it is considered as a match. The ratio is calculated using the Equation (1). All experimental codes with the dataset are available in the following public GitHub repository: https://github.com/ping543f/pdf-extraction-tool-comparison. Resources are provided upon request to the corresponding author.

$$R = 2 * M/T \tag{1}$$

where R is the matching ratio value of two sentences, M are the matches found among two sentences and T is the total number of elements in both sentences.

3.3. Experimental Setup

Python version 3.7.3 64 bit conda base is used for the development of the experimental program. The operating system used is macOS Big Sur version 11.5. The hardware configuration used is a 1.2 GHz dual core Intel Core m5 processor with 8 GB RAM.

4. Results and Discussion

As mentioned earlier, this study has three distinct goals: to find a better Python package for extracting text from PDF files, to find a better NLP framework for sentence boundary detection, and to evaluate the performance of PDF extraction packages in terms of memory footprint and runtime.

For this experiment, a dataset consisting of 10 scientific articles on the topic of Electric Double Layer Capacitors from different journal publishers is used. All the articles are downloaded in PDF format and converted into text using various popular Python PDF packages, namely Pypdf2, Pymupdf, Pdfminer.six, Pdftotext, Tika, and Grobid. Then, sentence boundary identification is applied to the extracted PDF documents using three different NLP packages, namely NLTK, Spacy, and Gensim. Then, the correctly identified sentences are matched with the ground truth file using the difflib package.

For the sentence boundary identification experiment, precision, recall and f-1 values are used as performance measures. The precision, recall and f-1 values are calculated using Equations (2)–(4), respectively.

$$Precision = \frac{Sentence_{correct}}{Sentence_{extract}} \tag{2}$$

$$Recall = \frac{Sentence_{correct}}{Sentence_{standard}} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

Here, $Sentence_{correct}$ stands for the correctly identified sentences from the extracted sentences extracted from the documents. On the other hand, $Sentence_{extract}$ stands for the total extracted sentences from the documents. $Sentence_{standard}$ stands for the total number of sentences in the ground truth file. Table 3 represents the comparison results of precision, recall, and f-1 values obtained in the experiment.

Table 3. Comparison score of precision, recall, and f-1 scores of different PDF-to-text extraction Python packages for different NLP frameworks.

Package	Precision			Recall			F-1		
	NLTK	Spacy	Gensim	NLTK	Spacy	Gensim	NLTK	Spacy	Gensim
Pypdf2	0.61	0.86	0.23	0.61	0.86	1.00	0.61	0.86	0.37
Pymupdf	0.63	0.70	0.18	1.00	1.00	1.00	0.75	0.82	0.30
Pdfminer.six	0.63	0.70	0.19	1.00	1.00	1.00	0.75	0.82	0.31
Pdftotext	0.62	0.69	0.27	1.00	1.00	1.00	0.74	0.81	0.42
Tika	0.63	0.70	0.20	1.00	1.00	1.00	0.75	0.82	0.75
Grobid	0.88	0.91	0.86	0.93	0.96	0.94	0.90	0.93	0.89

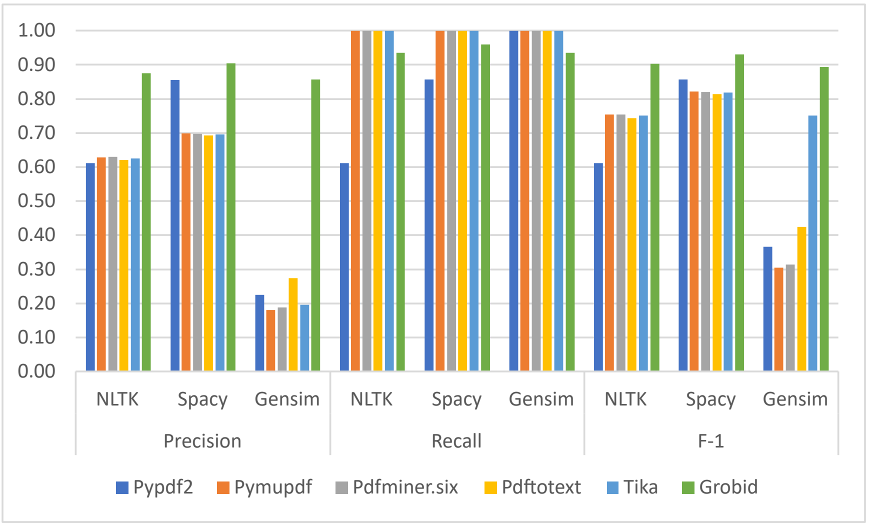

Figure 2. Precision, recall, and F-1 scores of different PDF-to-text extraction Python packages for different NLP frameworks.

Since three NLP frameworks are used to identify the sentence boundary, the experiment shows that the Spacy NLP framework always has higher precision and a higher f-1 score for each PDF-to-text extraction package than the other two frameworks, namely NLTK and Gensim. The PDF-to-text extraction package that achieved the highest precision and f-1 score is Grobid, which achieves 91% and 93% with the Spacy framework, respectively, while the NLTK and Gensim frameworks achieve 88% and 86% precision scores and 90% and 89% f-1 scores, respectively. This experimental result shows that the Grobid package performs better with Spacy in identifying complete sentences from the extracted plain text.

The precision and f-1 values of Grobid are also higher than the other five packages in all NLP frameworks.

Table 4 shows the match ratio results for all documents for all PDF-to-text conversion tools. The match ratio value is calculated using the Equation (1). The match ratio is calculated by comparing the documents converted by all the tools with the base documents, one by one. From the results, it can be seen that the Grobid tool has achieved the best average match ratio, 81%, when extracting text from PDF documents. This means that, on average, 81% of the text extracted by Grobid matched the original text created manually by the authors for the 10 documents used in this study. After Grobid, Pymupdf achieved a 57% match, which is significantly lower than the score achieved by Grobid. The performance of text extraction by the Grobid tool is also evident from the individual results for all documents. For each document, Grobid achieved a much better match rate than the other tools with which it was compared. In addition to sentence boundary extraction in conjunction with NLP frameworks, Grobid also performs better in pure PDF- to-text conversion.

Table 4. Match ratio values for different PDF extraction tools.

Document	Grobid	Pdfminer.six	Pdftotext	Pymupdf	Pypdf2	Tika
P1	0.92	0.76	0.19	0.84	0.19	0.79
P2	0.72	0.21	0.10	0.22	0.10	0.22
P3	0.87	0.64	0.23	0.69	0.13	0.66
P4	0.88	0.66	0.30	0.79	0.24	0.75
P5	0.61	0.11	0.05	0.12	0.04	0.14
P6	0.90	0.63	0.21	0.74	0.23	0.78
P7	0.63	0.12	0.05	0.12	0.07	0.18
P8	0.92	0.76	0.26	0.84	0.22	0.62
P9	0.89	0.71	0.21	0.81	0.26	0.74
P10	0.81	0.44	0.23	0.53	0.08	0.55
Average score	0.81	0.50	0.18	0.57	0.16	0.54

In [27], the authors compared four industrial optical character recognition programs for text content extraction with the character recognition-based measure, but they did not experiment with sentence boundary identification. This study showed good performance in character recognition, but was not promising in sentence boundary identification. Bast et al. [21] compared 14 PDF tools for text extraction and measured their performance on paragraph boundaries, distinguishing body text from non-body text, word boundaries, and reading sequences. They did not perform the sentence boundary extraction task. However, Grobid performed better than the other tools in identifying paragraph boundaries. This trend is also reflected in our study, where Grobid performed better than other tools in converting PDF-to-text and extracting sentence boundaries using the spacy nlp framework. The PDF2LaTex [26] tool proposed by Zelun et al. achieved 81% accuracy in converting mathematical expressions and text. However, from our experimental results, the Grobid tool performs better than PDF2LaTex in terms of accuracy, and the results with the NLTK, Spacy, and Gensim frameworks are 88%, 91%, and 86%, respectively. In [30,33], the authors experimented with extracting metadata, including titles and section headings. In both studies, Grobid was found to perform better in converting PDF-to-text compared to other tools. In addition, the f-1 values from these studies are almost identical to our experimental results, suggesting that Grobid performs better than other tools in other studies for different domains' data.

Considering the discussions, it can be deduced that the Grobid package can be used for better text extraction from scientific PDF documents than the other five packages and that the NLP package Spacy can be used for better identification of complete sentences from the extracted text than NLTK and Gensim. Figure 2 shows the comparison results for different performance metrics obtained for different PDF text extraction packages by NLP packages.

Another goal of this study is to determine the memory requirements and runtime of all packages used to extract PDF-to-text. Tables 5 and 6 show the memory usage and runtime results for all packages used in this study for the 10 ground truth documents.

Table 5. Memory space usage in Mega Bytes of different PDF-to-text extraction packages (a lower score is better).

Package	Memory Usage (MB)
Tika	159.46
Pypdf2	158.91
Pymupdf	156.37
Pdfminer.six	151.34
Pdftotext	150.75
Grobid	46.13

Table 6. Runtimes in second of different PDF-to-text extraction packages (a lower score is better).

Package	Runtime (Second)
Tika	3.35
Pypdf2	10.9
Pymupdf	1.23
Pdfminer.six	26.1
Pdftotext	1.96
Grobid	36.5

From Table 6, it can be seen that among all the PDF-to-text extraction packages, Pymupdf takes the least runtime of 1.23 s to extract text from 10 PDF documents, while Grobid has the highest runtime of all the compared packages at 36.5 s. On the other hand, Grobid requires the least memory with 46.13 megabytes, and Tika requires the highest memory with 159.46 megabytes among all the compared packages as shown in Table 5. Figures 3 and 4 show the visual representation of the memory usage and runtime of all compared packages. The runtime result is also supported by the work described in [21]. From the result presented in [21], Grobid took 42 s on average to convert a single file, and Pdftotext took 0.3 s per file. From our experimental result, we also find that Grobid takes the most time to convert 10 files, 3.65 s on average for a single file, and Pdftotext takes 0.196 s on average per file.

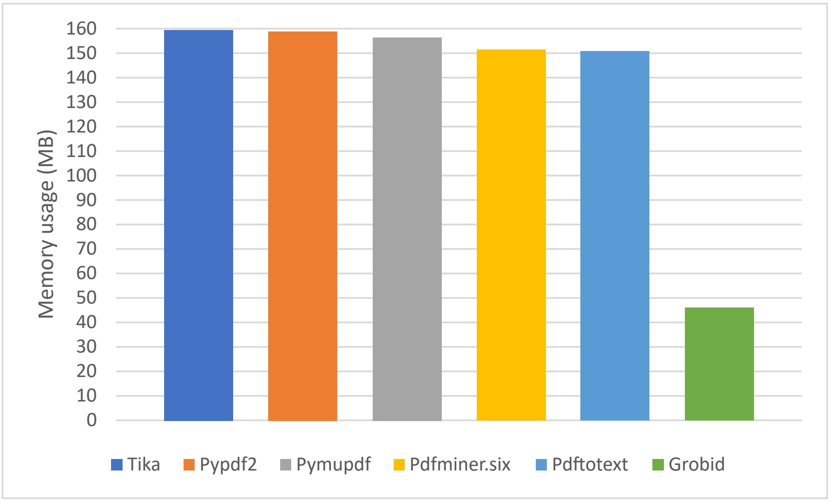

Figure 3. Comparison results of memory usage in Mega Bytes of different PDF-to-text extraction packages (a lower score is better).

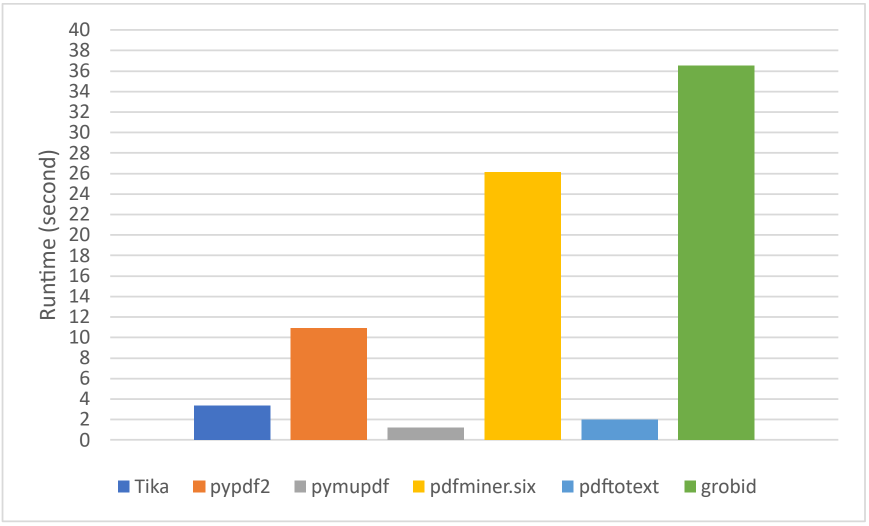

Figure 4. Comparison results of run times in seconds of different PDF-to-text extraction packages (a lower score is better).

To analyze the trend of memory usage and runtime of the PDF-to-text extraction packages, runtime and memory usage are calculated for 50 and 100 documents. The results are shown in Tables 7 and 8 for runtime and memory usage, respectively. The calculation results show that runtime and memory consumption for 50 and 100 documents follow a similar trend as for 10 documents. The Pymupdf package requires the least runtime while Pdfminer.six requires the highest runtime for converting text from PDF documents for 50 and 100 documents. Figure 5 presents the runtime comparison for all packages. On the other hand, Grobid consumes the least memory, while Tika requires the highest memory for converting text from PDF documents for both 50 and 100 documents, as shown in Figure 6. The experiment with 50 and 100 documents is performed in the same hardware and software setup mentioned in Section 3.3.

Table 7. Memory usage comparison for 10, 50, and 100 documents while converting them to text using different PDF-to-text extraction packages (a lower score is better).

Package	Memory Usage (MB) 10 Documents	Memory Usage (MB) 50 Documents	Memory Usage (MB) 100 Documents
Tika	159.46	200.52	232.43
Pypdf2	158.91	198.23	225.67
Pymupdf	156.37	200.01	220.45
Pdfminer.six	151.34	175.54	198.23
Pdftotext	150.75	180.07	222.97
Grobid	46.13	68.16	75.05

Table 8. Runtime comparison for 10, 50, and 100 documents while converting them to text using different PDF-to-text extraction packages (a lower score is better).

Package	Runtime (Seconds) 10 Documents	Runtime (Seconds) 50 Documents	Runtime (Seconds) 100 Documents
Tika	3.35	13.6	30.16
pypdf2	10.9	28.5	60.6
pymupdf	1.23	10.5	22.7
pdfminer.six	26.1	58.1	70.9
pdftotext	1.96	6.8	18.2
grobid	36.5	46.7	62.9

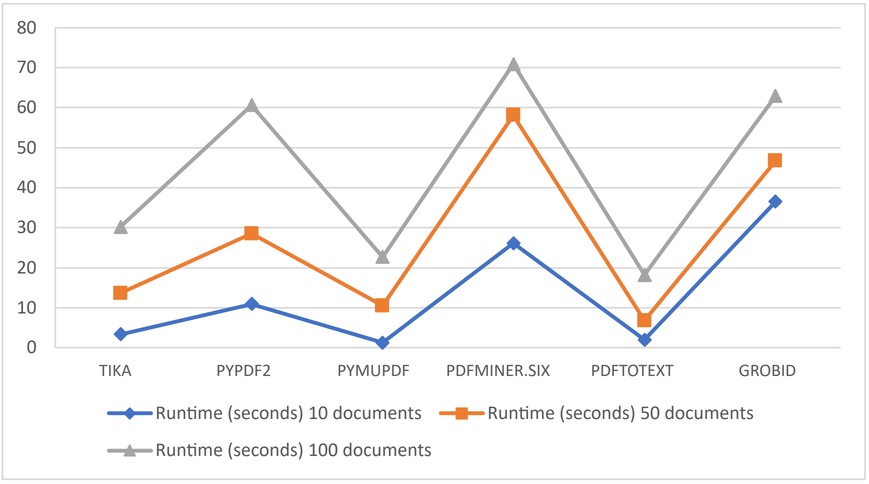

Figure 5. Comparison results of run times in seconds of different PDF-to-text extraction packages for 10, 50, and 100 documents (a lower score is better).

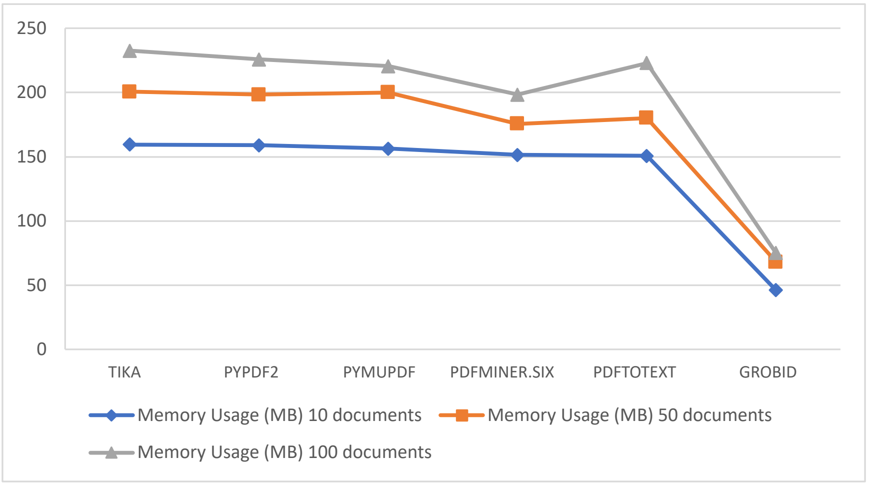

Figure 6. Comparison results of memory usage in Mega Bytes of different PDF-to-text extraction packages for 10, 50, and 100 documents (a lower score is better).

Following this analysis, and given the precision and f-1 values obtained, it is proposed to use Grobid for text extraction from scientific PDF documents. Moreover, the memory consumption of Grobid in converting text content from PDF documents is the lowest among all the compared packages, and Grobid also takes less time in converting a larger number of documents. This is evident from Table 8 and Figure 5. For converting 10 documents, Grobid takes the highest runtime of 36.5 s. However, when the number of documents is increased to 50 and 100, the Pdfminer.six package takes the highest runtime to complete the conversion process, and Grobid falls behind the Pdfminer.six package.

5. Conclusions

This article contains a detailed comparison of different Python packages for PDF-to-text extraction with their runtime and memory requirements for documents from the EDLC domain. It also presents the precision measure of correctly identified sentences from the extracted texts using three widely used natural language frameworks, namely NLTK, Spacy, and Gensim. The Grobid PDF-to-text extraction package achieved the highest f-1 score of 93% using NLP framework Spacy, while NLTK and Gensim achieved 88% and 86% precision values, respectively. Grobid also requires the least amount of memory to convert PDF documents to text. This study makes it easy for researchers working with text processing from the EDLC domain to choose the best combination of tools and techniques for extracting text with correct sentence boundaries from PDF documents. With the help of this work, EDLC researchers would waste less time in selecting appropriate tools and techniques for extracting sentence-level text from PDF documents. This study is also of great help in developing automated EDLC related information systems that need to extract text from PDF documents and use sequence labels, for example, an automated chemical information system that features automatic entity extraction elated to EDLC from the scientific literature. For this type of system, the text data needs to be as accurate as possible, and in this scenario, the current work can be very useful in extracting and creating clean text inputs from PDF documents. This study has been carried out using documents related to Electric Double Layer Capacitors. However, this study can also be carried out using scientific documents from other fields. The results can be analyzed for new insights, including domain-specific precision, recall, and f1 values, as well as differences in runtime, memory usage, and usage of other computational resources, including CPU utilization, thermal impact, and power consumption.

Author Contributions: Conceptualization, M.S.U.M. and J.S.; methodology, M.S.U.M. and J.S.; software, M.S.U.M. and T.B.S.; validation, K.Z.Z. and R.J.; formal analysis, M.S.U.M.; investigation, T.B.S.; resources, A.N. and F.A.; data curation, M.S.U.M.; writing—original draft preparation, M.S.U.M.; writing—review and editing, J.S. and R.J.; visualization, M.S.U.M.; supervision, J.S.; project administration, K.Z.Z.; funding acquisition, A.N. and F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Trans-Disciplinary Research Grant Scheme of the Ministry of Higher Education in Malaysia entitled: An Artificial Neural Network Sine Cosine Algorithm-based Hybrid Prediction Model for the Production of Cellulose Nanocrystals from Oil Palm Empty Fruit Bunch TRGS/1/2018/ UMP /01/1/4 (RDU191801-4), RDU182201 and The APC was fully funded by the Department of Computer Science, School of Systems and Technology, University of Management and Technology, Lahore 54782, Pakistan.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author or by opening an issue at https://github.com/ping543f/pdf-extraction-tool-comparison repository. The data are not publicly available due to the total implementation of the information system is still in progress.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. MacCartney, B. Understanding natural language understanding. In Proceedings of the ACM SIGAI Bay Area Chapter Inaugural Meeting, San Mateo, CA, USA, 16 July 2014.
2. Miah, M.; Sulaiman, J.; Sarwar, T.B.; Zamli, K.Z.; Jose, R. Study of Keyword Extraction Techniques for Electric Double-Layer Capacitor Domain Using Text Similarity Indexes: An Experimental Analysis. *Complexity* **2021**, 2021, 8192320 . [CrossRef]
3. Max Ved. NLP vs. NLU: From Understanding a Language to Its Processing—Data Science Central. 2021. Available online: http://bit.do/nlp-vs-nlu (accessed on 24 November 2021).
4. Jose, R.; Ramakrishna, S. Materials 4.0: Materials big data enabled materials discovery. *Appl. Mater. Today* **2018**, *10*, 127–132. [CrossRef]
5. Kim, E.; Huang, K.; Saunders, A.; McCallum, A.; Ceder, G.; Olivetti, E. Materials Synthesis Insights from Scientific Literature via Text Extraction and Machine Learning. *Chem. Mater.* **2017**, *29*, 9436–9444. [CrossRef]
6. Swain, M.C.; Cole, J.M. ChemDataExtractor: A Toolkit for Automated Extraction of Chemical Information from the Scientific Literature. *J. Chem. Inf. Model.* **2016**, *56*, 1894–1904. [CrossRef] [PubMed]
7. Honnibal, M.; Montani, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *Appear* **2017**, *7*, 411–420.
8. Pierre Carbonnelle. PYPL PopularitY of Programming Language Index. 2021. Available online: https://pypl.github.io/PYPL.html (accessed on 14 July 2021).
9. Kononova, O.; Huo, H.; He, T.; Rong, Z.; Botari, T.; Sun, W.; Tshitoyan, V.; Ceder, G. Text-Mined Dataset of Inorganic Materials Synthesis Recipes. *Sci. Data* **2019**, *6*, 203. [CrossRef] [PubMed]

10. Friedrich, A.; Heike, A.; Federico, T.; Johannes, H.; Renou, B.; Anika, M.; Lukas, L. The SOFC-exp corpus and neural approaches to information extraction in the materials science domain. *arXiv* **2020**, arXiv:2006.03039.
11. Hiszpanski, A.M.; Gallagher, B.; Chellappan, K.; Li, P.; Liu, S.; Kim, H.; Han, J.; Kailkhura, B.; Buttler, D.J.; Han, T.Y.J. Nanomaterial Synthesis Insights from Machine Learning of Scientific Articles by Extracting, Structuring, and Visualizing Knowledge. *J. Chem. Inf. Model.* **2020**, *60*, 2876–2887. [CrossRef] [PubMed]
12. Kuniyoshi F.; Kohei M.; Jun O.; Makoto M. Annotating and extracting synthesis process of all-solid-state batteries from scientific literature. *arXiv* **2020**, arXiv:2002.07339.
13. Sarwar, T.B.; Noor, N.M.; Miah, M.S.U.; Rashid, M.; Al Farid, F.; Husen, M.N. Recommending Research Articles: A Multi-Level Chronological Learning-Based Approach Using Unsupervised Keyphrase Extraction and Lexical Similarity Calculation. *IEEE Access* **2021**, *9*, 160797–160811. [CrossRef]
14. Miah, M.S.U.; Sulaiman, J.; Azad, S.; Zamli, K.Z.; Jose, R. Comparison of document similarity algorithms in extracting document keywords from an academic paper. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021; pp. 631–636.
15. Nadkarni, P.M.; Ohno-Machado, L.; Chapman, W.W. Natural language processing: An introduction. *J. Am. Med. Inform. Assoc.* **2011**, *18*, 544–551. [CrossRef] [PubMed]
16. Miah, M.S.U.; Tahsin, M.S.; Azad, S.; Rabby, G.; Islam, M.S.; Uddin, S.; Masuduzzaman, M. A Geofencing-based Recent Trends Identification from Twitter Data. In *IOP Conference Series: Materials Science and Engineering*; Institute of Physics Publishing: Bristol, UK , 2020; Volume 769, p. 012008.
17. Sarwar, T.B.; Noor, N.M. An experimental comparison of unsupervised keyphrase extraction techniques for extracting significant information from scientific research articles. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021; pp. 130–135.
18. Miah, M.S.U.; Bhowmik, A.; Anannya, R.T. Location, context and device aware framework (LCDF): A unified framework for mobile data management. In Proceedings of the International Conference on Computing Advancements, Dhaka, Bangladesh, 10–12 January 2020; ACM International Conference Proceeding Series; Association for Computing Machinery: New York, NY, USA, 2020.
19. Michaud A.; Oliver A.; Trevor A. C.; Graham N.; Severine G. Integrating automatic transcription into the language documentation workflow: Experiments with Na data and the Persephone toolkit. *Lang. Doc. Conserv.* **2018**, *12*, 393–429.
20. Friedrich, F.; Mendling, J.; Puhlmann, F. Process model generation from natural language text. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer, Berlin/Heidelberg, Germany, 2011; Volume 6741, pp. 482–496.
21. Bast, H.; Korzen, C. A Benchmark and Evaluation for Text Extraction from PDF. In Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, Toronto, ON, Canada, 19–23 June 2017; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2017.
22. Taskaya, B. Reiz: Structural Source Code Search. *J. Open Source Softw.* **2021**, *6*, 3296. [CrossRef]
23. Taskaya, B. Top PyPI Packages: A Monthly Dump of the 5000 Most-Downloaded Packages from PyPI. 2021. Available online: https://hugovk.github.io/top-pypi-packages/ (accessed on 24 November 2021).
24. Azimjonov, J.; Alikhanov, J. Rule Based Metadata Extraction Framework from Academic Articles. *arXiv* **2018**, arXiv:1807.09009.
25. Nasar, Z.; Jaffry, S.W.; Malik, M.K. *Information Extraction from Scientific Articles: A Survey*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; Volume 117.
26. Wang, Z.; Liu, J.C. PDF2LaTeX: A Deep Learning System to Convert Mathematical Documents from PDF to LaTeX. In Proceedings of the ACM Symposium on Document Engineering, DocEng 2020, San Jose, CA, USA, 29 September–1 October 2020.
27. Nadiah, A.; Rahman, C.A.; Abdullah, H.; Zainuddin, S.; Jaludin, A. The Comparisons of Ocr Tools: A Conversion Case in the Malaysian Hansard Corpus Development. *Malays. J. Comput.* **2019**, *4*, 335–348.
28. Duretec, K.; Rauber, A.; Becker, C. A Text Extraction Software Benchmark Based on a Synthesized Dataset. In Proceedings of the 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL), Toronto, ON, Canada, 19–23 June 2017; pp. 1–10. [CrossRef]
29. McKie, J.X. PyMuPDF 1.19.4 Performance Evaluation. 2021. Available online: https://pymupdf.readthedocs.io/en/latest/app2.html (accessed on 22 October 2021).
30. Mario, L.; Yao, K.; Breitinger, C.; Beel, J.; Gipp, B. Evaluation of header metadata extraction approaches and tools for scientific PDF documents. In Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries, Indianapolis, IN, USA, 22–26 July 2013; pp. 385–386.
31. Patrice Lopez. GROBID. 2021. Available online: https://github.com/kermitt2/grobid (accessed on 22 October 2021).
32. Palmer, J.A. Pdftotext · PyPI. 2017. Available online: https://pypi.org/project/pdftotext/ (accessed on 20 October 2021).
33. Singh, M.; Barua, B.; Palod, P.; Garg, M.; Satapathy, S.; Bushi, S.; Ayush, K.; Rohith, K.S.; Gamidi, T.; Goyal, P. OCR++: A Robust Framework for Information Extraction from Scholarly Articles. *arXiv* **2016**, arXiv:1609.06423
34. George, S. Sentence boundary detection in legal text. In Proceedings of the Natural Legal Language Processing Workshop, Minneapolis, MN, USA, 7 June 2019; pp. 31–38.

35. Thiengburanathum, P. A Comparison of Thai Sentence Boundary Detection Approaches Using Online Product Review Data. In Proceedings of the International Conference on Network-Based Information Systems, Victoria, BC, Canada, 31 August–2 September 2020; Springer: Cham, Switzerland, 2020; pp. 405–412.
36. Tshitoyan, V.; Dagdelen, J.; Weston, L.; Dunn, A.; Rong, Z.; Kononova, O.; Persson, K.A.; Ceder, G.; Jain, A. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature* **2019**, *571*, 95–98. Available online: http://dx.doi.org/10.1038/s41586-019-1335-8 (accessed on 19 September 2021). [CrossRef] [PubMed]
37. Guha, S.; Mullick, A.; Agrawal, J.; Ram, S.; Ghui, S.; Lee, S.C.; Bhattacharjee, S.; Goyal, P. MatScIE: An automated tool for the generation of databases of methods and parameters used in the computational materials science literature. *Comput. Mater. Sci.* **2021**, *192*, 110325. [CrossRef]
38. Olivetti, E.A.; Cole, J.M.; Kim, E.; Kononova, O.; Ceder, G.; Han, T.Y.J.; Hiszpanski, A.M. Data-driven materials research enabled by natural language processing and information extraction. *Appl. Phys. Rev.* **2020**, *7*, 04131. [CrossRef]
39. Phaseit Inc.; Fenniak, M. PyPDF2 Documentation—PyPDF2 1.26.0 Documentation. 2016. Available online: https://pythonhosted.org/PyPDF2/ (accessed on 29 November 2021).
40. Marsman, P.; Shinyama, Y.; Guglielmetti, P. Pdfminer.six 20201018 Documentation. 2019. Available online: https://pdfminersix.readthedocs.io/en/latest/ (accessed on 27 November 2021).
41. McKie, J.X. PyMuPDF Documentation—PyMuPDF 1.18.15 Documentation. 2015. Available online: https://pymupdf.readthedocs.io/en/latest/ (accessed on 27 November 2021).
42. Fenniak, M. pyPdf. 2021. Available online: http://pybrary.net/pyPdf/ (accessed on 29 November 2021).
43. Shinyama, Y. Pdfminer. 2021. Available online: https://github.com/euske/pdfminer (accessed on 28 November 2021).
44. Shinyama, Y.; Guglielmetti, P.; Marsman, P. Converting a PDF File to Text—Pdfminer.six. 2019. Available online: https://pdfminersix.readthedocs.io/en/latest/topic/converting_pdf_to_text.html (accessed on 28 November 2021).
45. Artifex Sofware Inc. MuPDF. 2020. Available online: https://www.mupdf.com/ (accessed on 30 November 2021).
46. Kristian Høgsberg. Poppler. 2021. Available online: https://poppler.freedesktop.org/ (accessed on 30 November 2021).
47. Glyph & Cog LLC. xpdf. 2021. Available online: http://www.xpdfreader.com/contact.html (accessed on 30 November 2021).
48. Apache Software Foundation. Apache Tika—Getting Started with Apache Tika. 2021. Available online: https://tika.apache.org/1.27/gettingstarted.html (accessed on 1 December 2021).
49. NLTK. NLTK Corpora. 2021. Available online: http://www.nltk.org/nltk_data/ (accessed on 15 August 2021).
50. Bird, S.; Loper, E.; Klein, E. *Natural Language Processing with Python*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2009.
51. Řehůřek, R.; Sojka, P. Software Framework for Topic Modelling with Large Corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, 22 May 2010 ; European Language Resources Association: Paris, France, 2010 ; pp. 45–50. Available online: http://is.muni.cz/publication/884893/en (accessed on 4 September 2021).
52. Prabhakaran, S. Gensim Tutorial—A Complete Beginners Guide. 2018. Available online: https://www.machinelearningplus.com/nlp/gensim-tutorial/ (accessed on 1 December 2021).
53. Explosion. Sentencizer. 2021. Available online: https://spacy.io/api/sentencizer (accessed on 1 December 2021).
54. NLTK Project. 2021. Natural Language Toolkit—NLTK 3.6.2 Documentation. Available online: http://www.nltk.org/ (accessed on 3 December 2021).
55. Řehůřek, R. Documentation—Gensim. 2021. Available online: https://radimrehurek.com/gensim/auto_examples/index.html#documentation (accessed on 1 December 2021).
56. Spacy.io. Install spaCy · spaCy Usage Documentation. 2021. Available online: https://spacy.io/usage (accessed on 3 December 2021).
57. Python Software Foundation. Difflib-Helpers for Computing Deltas. 2021. Available online: https://docs.python.org/3/library/difflib.html (accessed on 4 December 2021).