




pygetpapers: a Python library for automated retrieval of scientific literature

Ayush Garg ¹, Richard D Smith-Unna ², and Peter Murray-Rust ³

¹ University of Richmond, Richmond, VA 23173, United States ² LabDAO, <https://www.labdao.com/>
³ Yusuf-Hamied Department of Chemistry, University of Cambridge, Lensfield Road, Cambridge, CB2 1EW, UK

DOI: [10.21105/joss.04451](https://doi.org/10.21105/joss.04451)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Chris Hartgerink](#)  

Reviewers:

- [@khinsen](#)
- [@VeroniqueLegrand](#)
- [@kjappelbaum](#)

Submitted: 24 May 2022

Published: 07 July 2022

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

pygetpapers has been developed to allow searching of the scientific literature in repositories with a range of textual queries and metadata. It downloads content using APIs in an automated fashion and is designed to be extensible to the growing number of Open Access repositories.

Statement of Need

An increasing amount of research, particularly in medicine and applied science, is now based on meta-analysis and systematic review of the existing literature (“[Systematic Reviews](#),” 2022). In such reviews, scientists frequently download thousands of articles and analyze them by Natural Language Processing (NLP) through Text and Data Mining (TDM) or Content Mining. A common approach is to search bibliographic resources with keywords, download the hits, scan them manually, and reject papers that do not fit the criteria for the meta-analysis. The typical text-based searches on sites are broad, with many false positives and often only based on abstracts. We know of cases where systematic reviewers downloaded 30,000 articles and eventually used 30. Retrieval is often done by crawling/scraping sites, such as journals but is easier and faster when these articles are in Open Access repositories such as arXiv, EuropePMC, bioRxiv, medRxiv. But each repository has its own API and functionality, which makes it hard for individuals to (a) access, (b) set flags, and (c) use generic queries. In 2015, we reviewed tools for scraping websites and decided that none met our needs and so developed getpapers ([Smith-Unna, 2021](#)), with the key advance of integrating a query submission with bulk fulltext-download of all the hits.

pygetpapers

getpapers was written in NodeJS and has now been completely rewritten in Python3 (pygetpapers) for easier distribution and integration. Typical use of getpapers is shown in a recent paper ([Wind et al., 2021](#)) where the authors “analyzed key term frequency within 20,000 representatives [Antimicrobial Resistance] articles”.

An important aspect is to provide a simple cross-platform approach for scientists who may find tools like curl too complex and want a one-line command to combine the search, download, and analysis into a single: “please give me the results”. We’ve tested this on many interns who learn pygetpapers in minutes. It was also easy to wrap it into a tkinter graphical user interface (GUI) ([Lundh, 1999](#)). The architecture of the results is simple and natural, based on full-text files in the normal filesystem. The result of pygetpapers is interfaced using a “main” or “controller” JSON file (for eg. eupmc_results.json), which allows corpus to be reused/added to. This allows maximum flexibility of re-use and some projects have large amounts of derived data in these directories.

pygetpapers takes the approach of downloading once and re-analyzing later on local filestore. This saves repeated querying where connections are poor or where there is suspicion that publishers may surveil users. Moreover, publishers rarely provide more than full-text Boolean searches, whereas local tools can analyze sections and non-textual material.

We do not know of other tools which have the same functionality. curl (Hostetter et al., 1997) requires detailed knowledge of the download protocol. VosViewer (J. & L, 2010) is mainly aimed at bibliography/citations.

Overview of the architecture

Data

raw data

The download may be repository-dependent but usually contains:

- download metadata. (query, date, errors, etc.)
- journal/article metadata. We use JATS-NISO (*Standardized Markup for Journal Articles, 2021*) which is widely used by publishers and repository owners, especially in bioscience and medicine. There are over 200 tags.
- fulltext. This can be
 - XML (fulltext and metadata)
 - images (these may not always be available)
 - tables (these are often separate)
 - PDF - usually includes the whole material but not machine-sectioned
 - HTML . often available on websites
- supplemental data. This is very variable, often as PDF but also raw data files and sometimes zipped. It is not systematically arranged but pygetpapers allows for some heuristics.
- figures. This is not supported by some repositories, and others may require custom code.

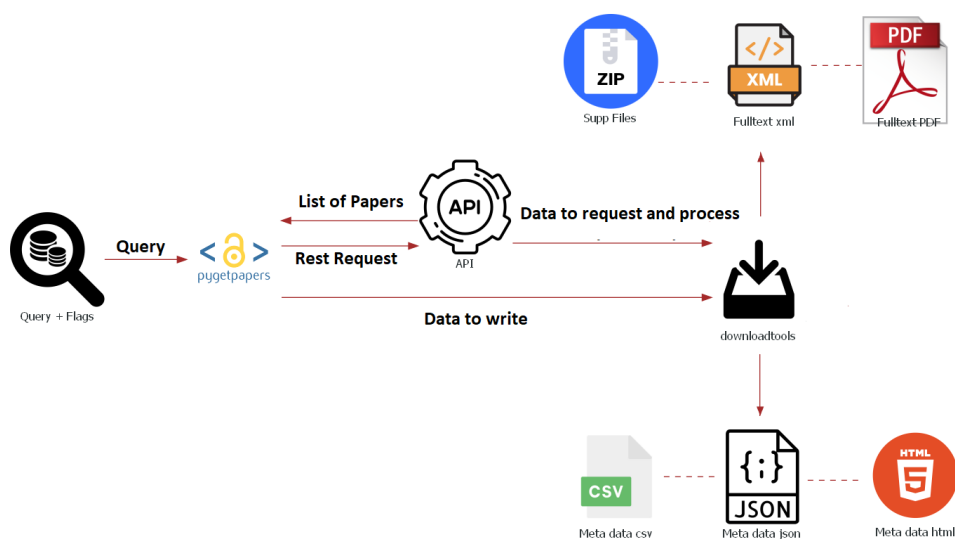


Figure 1: Fig.1 Architecture of pygetpapers

Fig.1 Architecture of pygetpapers

For this reason we create a directory structure with a root (`CProject`) and a (`CTree`) subdirectory for each downloaded article or document. `pygetpapers` will routinely populate this with 1-5 files or subdirectories (see above). At present `pygetpapers` always creates a `*_result.json` file (possibly empty) and this can be used as a marker for identifying `CTrees`. This means that a `CProject` contains subdirectories which may be `CTrees` or not, distinguished by this marker.

derived data

In addition to the downloaded data (already quite variable), users often wish to create new derived data and this directory structure is designed so that tools can add an arbitrary amount of new data, normally in sub-directory trees. For example we have sibling projects that add data to the `CTree`:

- `docanalysis` (text analysis including NLTK and `spaCy/sciSpaCy`)
- `pyamiimage` [image processing and analysis of figures](#)

```
C:.\
| eupmc_results.json
|
|---PMC8157994
|     eupmc_result.json
|     fulltext.xml
|
|---PMC8180188
|     eupmc_result.json
|     fulltext.xml
|
```

and with examples of derived data

```
|---PMC8198815
|     eupmc_result.json
|     fulltext.xml
|.     bag_of_words.txt
|.     figure/
|.         raw.jpg
|.         skeleton.png
|
|---10.9999_123456 # CTree due to fooRxiv_result.json
|     fooRxiv_result.json
|     fulltext.xml
|.     bag_of_words.txt
|.     search/
|.         results/
|.         terpenes.csv
|
|.     univ_bar_thesis_studies_on_lantana/ # CTree dues to thesis_12345_results.json
|.         thesis_12345_results.json
|.         fulltext.pdf
|.         figures/
|.             figure/
|.                 Fig1/
|.
|___summary/ # not CTree as no child *_results.json
|.     bag_of_words.txt
```

```
|.      figures/  
|      <aggregated and filtered figures>
```

Typical download directory

Several types of download have been combined in this CProject and some CTrees have derived data.

Code

Download protocol

Most repository APIs provide a cursor-based approach to querying:

1. A query is sent and the repository creates a list of M hits (pointers to documents), sets a cursor start, and returns this information to the pygetpapers client.
2. The client requests a chunk of size $N \leq M$ (normally 25-1000) and the repository replies with N pointers to documents.
3. The server response is pages of hits (metadata) as XML, normally ≤ 1000 hits per page (1 sec).
4. pygetpapers - incremental aggregates XML metadata as python dict in memory.
5. If cursor indicates next page, pygetpapers submits a query for next page, otherwise it terminates the data collection and processes the python dict.
6. If user has requested supplemental data (e.g., references, citations, fulltext) then the pygetpapers iterates through the python dict and uses the identifier, usually in the form of DOI, to query and download supplemental data separately.
7. When the search is finished, pygetpapers writes the metadata to CProject (Top level project directory) as JSON (total, and creates CTrees (per-article directories) with individual metadata).
8. It also recovers from crashes and restarts if needed).

The control module `pygetpapers.py` reads the commandline and

- Selects the repository-specific downloader
- Creates a query from user input and/or terms from dictionaries
- Adds options and constraints
- Downloads according to the protocol above, including recording progress in a metadata file

Generic downloading concerns

- Download speeds. Excessively rapid or voluminous downloads can overload servers and are sometimes hostile (DOS). We have discussed this with major sites (EuropePMC, biorXiv, Crossref, etc. and therefore choose to download sequentially instead of sending parallel requests in pygetpapers.
- Authentication (alerting repo to downloader header). pygetpapers supports anonymous, non-authenticated, access but includes a header (e.g., for Crossref)

Design

The tool has been designed for ease of implementation, installation (including platform independence) and future extension. It also abstracts some of the variation in query languages and APIs (where there do not appear to be standards). For example for “date”, EuropePMC uses `FIRST_PDATE[DD-MM-YYYY to DD-MM-YY]` (This is the format in which you provide a date constraint to a query for EuropePMC) but bioRxiv uses `DD-MM-YYYY/DD-MM-YY`. `pygetpapers` provides `DATE` as an abstraction. It also uses a commandline, which makes it easy either to wrap the use in system calls, or layer a GUI on top.

Some repositories only support metadata while others include text and some even provide links to data downloads; again `pygetpapers` supports this range. Because there are hundreds of repositories (including preprints) the design includes a modular approach. And because some repositories emit variable amounts of information we can customise the outputs.

Implementation

`getpapers` was implemented in NodeJS, which allows multithreading and therefore potentially download rates of several XML documents per second on a fast line. Installing NodeJS was a problem on some systems (especially Windows) and was not well suited for integration with scientific libraries (mainly coded in Java and Python). We, therefore, decided to rewrite in Python, keeping only the command line and output structure, and have found very easy integration with other tools, including GUIs. `pygetpapers` can be run both as a command-line tool and a module, which makes it versatile.

core

The core mainly consists of:

- `pygetpapers.py` (query-builder and runner). This includes query abstractions such as dates and Boolean queries for terms
- `download_tools.py` (generic code for query/download (REST))

repository interfaces

We have tried to minimise the amount of repository-specific code, choosing to use declarative configuration files. To add a new repository you will need to:

- create a configuration file (Fig. 2)
- subclass the repo from `repository_interface.py`
- add any repository specific code to add features or disable others

Interface with other tools

Downloading is naturally modular, rather slow, and we interface by writing all output to the filesystem. This means that a wide range of tools (Unix, Windows, Java, Python, etc.) can analyze and transform it. The target documents are usually static so downloads only need to be done once. Among our own downstream tools are

- `pyami` (Murray-Rust, 2021) - sectioning the document
- `docanalysis` (Ayush Garg, 2021) - textual analysis and Natural Language Processing
- `pyamiimage` (Peter Murray-Rust, 2021) - analysis of the content of images in downloaded documents

- third party text analysis of PDF using GROBID([GROBID, 2008--2021](#)) and PDF-Box([Apache PDFBox® - a Java PDF Library, n.d.](#)).

Acknowledgements

We acknowledge contributions from Shweata N. Hegde in helping write the documentation. We also acknowledge Matthew Evans' support to help improve the quality of the code, and the repository.

Contribution statement

Ayush Garg: Development of the Tool, Architecture. Richard D. Smith-Unna: Base framework, work on getpapers (predecessor to pygetpapers). Peter Murray-Rust: Supervision, framework, writing manuscript.

References

- Apache PDFBox® - a java PDF library.* (n.d.). <https://pdfbox.apache.org/>
- Ayush Garg, S. N. H. (2021). *Docanalysis - unsupervised entity extraction* (Version 0.0.7) [Computer software]. <https://github.com/petermr/docanalysis>
- GROBID.* (2008--2021). GitHub. <https://github.com/kermitt2/grobid>
- Hostetter, M., Kranz, D. A., Seed, C., Terman, C., & Ward, S. (1997). Curl: A gentle slope language for the web. *World Wide Web Journal*, 2(2), 121–134.
- J., E. N., & L, W. (2010). Software survey: VOSviewer, a computer program for bibliometric mapping'. *Scientometrics*, 84(2). <https://doi.org/10.1007/s11192-009-0146-3>
- Lundh, F. (1999). An introduction to tkinter. URL: [www. Pythonware. Com/Library/Tkinter/Introduction/Index. Htm](http://www.pythonware.com/Library/Tkinter/Introduction/Index.Htm).
- Murray-Rust, P. (2021). *Pyami - semantic reader of the scientific literature* (Version 0.0.17) [Computer software]. <https://github.com/petermr/pyami/>
- Peter Murray-Rust, V. S. M. R., Anubhab Chakraborty. (2021). *Pyamiimage - tools to extract semantic information from scientific diagrams* (Version 0.0.8) [Computer software]. <https://github.com/petermr/pyamiimage>
- Smith-Unna, R. (2021). *Getpapers - tools to extract semantic information from scientific diagrams* (Version 0.4.17) [Computer software]. <https://github.com/ContentMine/getpapers>
- Standardized markup for journal articles: Journal article tag suite (JATS) | NISO website.* (2021). www.niso.org/standards-committees/jats .
- Systematic reviews. (2022). *BioMed Central*.
- Wind, L. L., Briganti, J. S., & Brown, A. M. (2021). Finding what is inaccessible: Antimicrobial resistance language use among the one health domains. *antibiotics*. <https://doi.org/10.3390/antibiotics10040385>