FabricETL GitHub Repository Outline

Contents:

1. Repository Structure Outline

The **Main** branch has no subdirectories. The top-level files are located here such as project specifications.

The **Document Image Analysis** branch has two subfolders: *OCR* and *Figure-to-Text Extraction*. This branch corresponds to the data extraction half of the project.

The **Research Corpus Generation** branch has no subdirectories. This branch corresponds to the materials aggregation half of the project.

Fabric (Repository Root)

├── Main
│
│
│
├── Document-Image-Analysis
│ ├── OCR
│ └── Figure-to-Text-Extraction
│
└── Research-Corpus-Generation

2. Reasoning for Repository Creation: Problem and Solution

The **problem** this project is trying to solve is finding what methods and materials have been shown to produce the best moisture-wicking and drying performance of apparel fabrics. Methods encompass material composition and application, while materials encompass base materials used. The **need** is aggregated, filtered, relevant research data, and automated extraction of relevant data from that research data. Examples of this need in terms of textile properties might include material or yarn density, while performance metrics might contain figures for drying time or wicking heights of applied fluids over a set time.
The **solution** determined to be best consisted of sourcing data with various functions to filter, download, fully-extract and clean data from those papers. The user inputs search query parameters and the program retrieves metadata of papers that match. It cleans and filters papers locally if a repository does not allow filtering at certain granularity while local filtering does. The first output is a XLSX file containing these metadata for papers that have search field matches in one and up to three research paper repositories. The next process downloads all papers listed in the XLSX in PDF format and again performs filtering by search terms if need be. The program then focuses on Document Image Analysis to extract data from figures from charts, tables, or text. This results in an Excel file which contains all the property and metric data desired.

```
FabricETL (orchestrator)
  │
  ├── ScholarSweep.py (import as module)
  │     └── API queries → filter → download PDFs → metadata.xlsx
  │
  └── TextileVision.py (import as module)
        └── OCR/F2T on PDFs → extract metrics → fabric_data.xlsx
```

3. Solution Architecture

a. Programming Languages Used

Python is easy to write and implement, and all tools used in the program are available in the Python ecosystem.

b. Resources

Search API:

CrossRef API: A public REST API that exposes the scholarly metadata deposited with Crossref, including DOIs, bibliographic details, funding data, license information, and links to full text where available.

Search Domain: Does NOT allow searching by exact phrases or searching of full-texts. If the user selects searching by either of those methods in the prompts, the filtering will be executed locally. Utilizes “best match”, which may or may not return many phrase matches. Does allow filtering by journal.

OpenAlex API: A free, open API over the OpenAlex index of the global research ecosystem, providing structured access to works, authors, venues, institutions, and concepts derived from sources such as Crossref, PubMed, and institutional repositories.

Search Domain: Allows searching by keyword or exact phrase, and in title, abstract, or full-text, and allows filtering by journal.

PubMed API: NCBI’s programmatic interface to PubMed and related databases, allowing users to search and retrieve biomedical citation and abstract records (and associated metadata) in a structured, machine-readable form.

Search Domain: Does NOT allow searching of full-texts or filtering by journal. If the user selects searching by either of those methods in the prompts, the filtering will be executed locally, nor does it allow

Requests: A popular, simple, and elegant Python library for making HTTP requests and interacting with web services and APIs. Needed to do API calls for the above three API’s.

Filter API:

UnPaywall API: Unpaywall is an open database and service that aggregates information about free-to-read (open access) versions of scholarly articles, harvested from publishers and repositories worldwide. It provides a REST API that lets users look up a DOI and retrieve metadata and links to any available open access copies

Bulk Downloader:

scihub.py: an unofficial API for Sci-hub. scihub.py can search for papers on Google Scholars and download papers from Sci-hub. It can be imported independently or used

from the command-line. It allows for bulk downloading of papers and is used when the paper is classified as non-OA (non-open access).

Excel Read/Write:

Openpyxl: The openpyxl package is a powerful, open-source Python library used for reading, writing, and modifying Excel 2010 and later files.

OCR Extraction:

Pdfplumber: The pdfplumber Python package is an open-source library used for efficiently extracting text, tables, and other detailed layout information from PDF files.

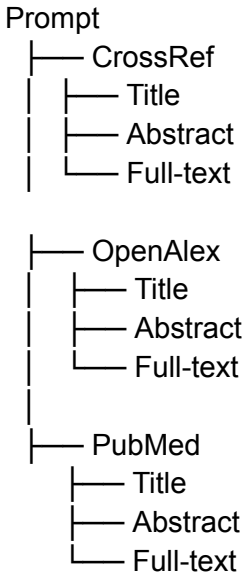
F2T (Figure-to-Text Extraction):

ChartVLM-L/B (used for charts)**:** Distinguishes itself from other Figure-to-Text (F2T) and Multi-modal Large Language Models (MLLMs) through a specific "perception-before-reasoning" philosophy and a modular architecture.
Unlike unified models that attempt to interpret a chart in one pass, ChartVLM separates the task into clear structural and cognitive stages. While models like MatCha or UniChart often use a single end-to-end transformer, ChartVLM uses two distinct decoders: a Base Decoder specifically handles Perception Tasks, such as converting chart pixels directly into CSV data, chart titles, and types. An Auxiliary Decoder is triggered only for Cognitive Tasks (e.g., complex reasoning or summarization). It uses the structural data from the Base Decoder as an "interpretable pattern" to ground its answers.

Table-LLaVA (used for tables): An open-source multimodal large language model (MLLM) specifically engineered to perceive and reason over table images. It extends the standard LLaVA (Large Language and Vision Assistant) architecture to handle complex tabular structures and data.

c. I/O

Inputs: User chooses how many parameters will be used to search across the repositories. You're first prompted which repository or repositories you want to query, and then asked what fields to use. Search terms are always entered in comma-separated format, with phrases simply entered in entirety. If a user enters "moisture wicking fabric, garment", it understands that to be the exact phrase "moisture wicking fabric" and the keyword "garment". All possible inputs are shown below in an ASCII diagram.



Outputs: 1. The first output is a XLSX file which represents a curated list of all the papers that matched the search terms. output which contains all the properties and metrics on the columns, and the specimen properties and metrics for all extracted specimens in the experiments of the research. Rows of papers which failed to download are turned red, while those that were successfully downloaded are turned green (if the download phase did not start, it will have no fill). The XLSX represents a curated list of papers which are believed to be relevant to the search query topic, and the user can reference at any time by navigating to the API Query folder which is created during every query in the same directory as the Python script.
2. A final dataset containing all fabric properties and moisture performance metrics available will also be generated in the same directory. A list of output columns here would number into the hundreds and thus isn't listed in this documentation, but can be accessed in the Main branch as file "MetricsFullList" and variations of that filename. The possible XLSX columns are *Study Title, Study Author(s), DOI, Abstract, OA Status* and *Journal Name*.The XLSX file is also split into two worksheets, *OA* and *Non-OA*. OA simply means *Open Access* and refers to if the paper is freely downloadable for the public without paying or needing credentials.

d. Methods

Several processes were implemented to help speed up runtime and most efficiently use memory, as well as allow resuming of the program if the user needs to kill it for whatever reason.

Multithreading: To speed up runtime, multithreading is used. If you choose to run queries on more than one repository, separate processes are used for each.

Low-Memory Mode: An option is present to run the program with what effectively is low-RAM logic; the results of the query are written to XLSX more quickly to free up RAM. By default, the program will begin to batch write query results once total results exceed 50,000 across all repositories being queried. This low-memory mode further allows you to stream results directly to the disk by writing every single result to disk as it's received.

Save: This feature allows the user to continue the program if they had to kill the process for some reason. If it was downloading metadata from repository API's, it will go back to doing that. If it was filtering metadata or searching papers locally, it will continue to do that. The user simply needs to enter /save and the program will write to itself the necessary data to resume on restart and then kill the process.

4. Program Process Flow and Details

The main process is named FabricETL, with ETL being a term meaning *Extract-Transform-Load* that describes extraction of data, transformation of that data, and then loading it into another format for use. ScholarSweep is the program which queries repository API's and filters and cleans all metadata, then writes this to an Excel file. TextileVision is an engine of OCR and F2T packages that extracts data from text, and elements like tables and charts.

Inputs include keywords and/or exact phrases for study title(s), abstract, or full-text, as well as journals to filter by. All of this is entered at the beginning by the user, and everything beyond that is done by FabricETL.

WARNINGS: if you open the output files before the process is done, it will throw an exception and tell you to close those files.

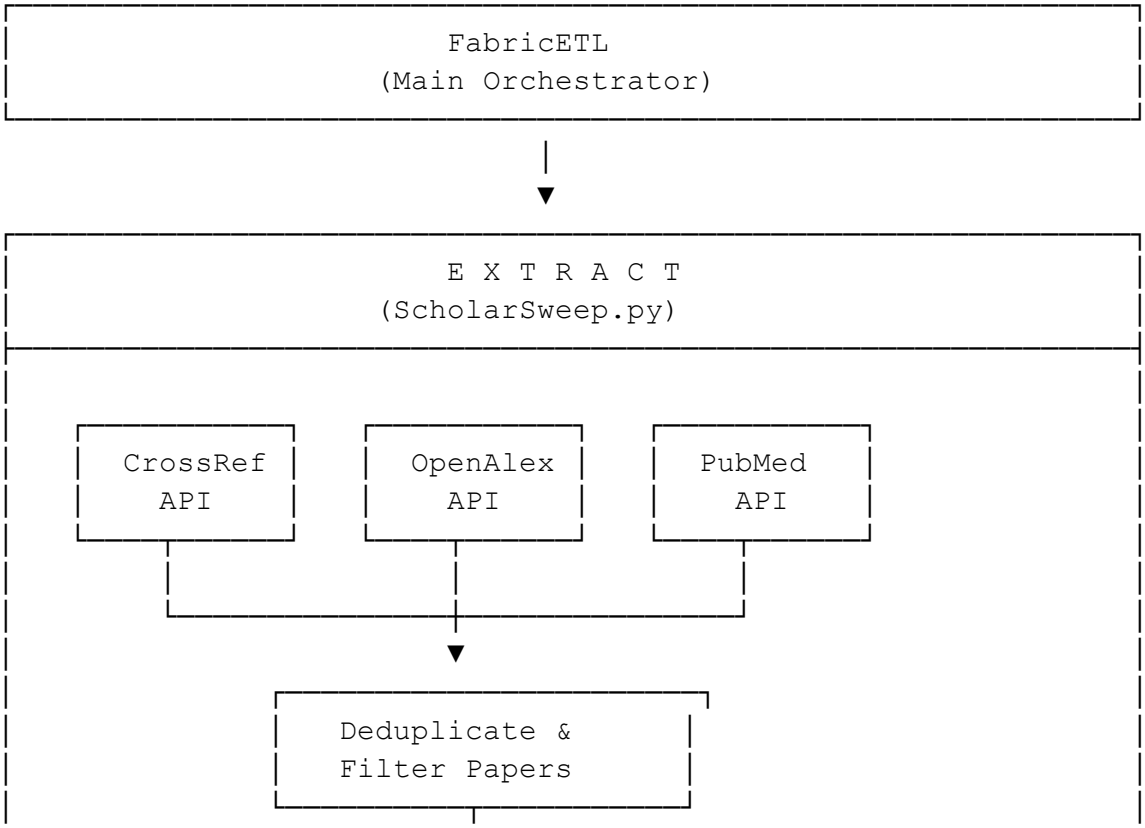
ScholarSweep
Filter millions of papers down to a few

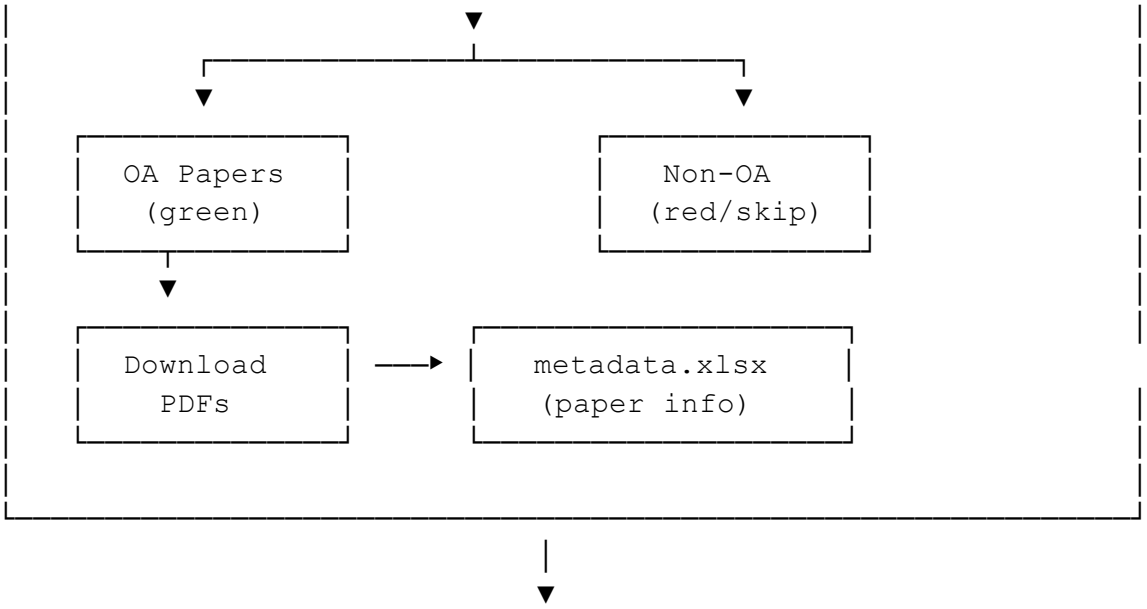
TextileVision
Convert PDFs into structured fabric properties and metrics

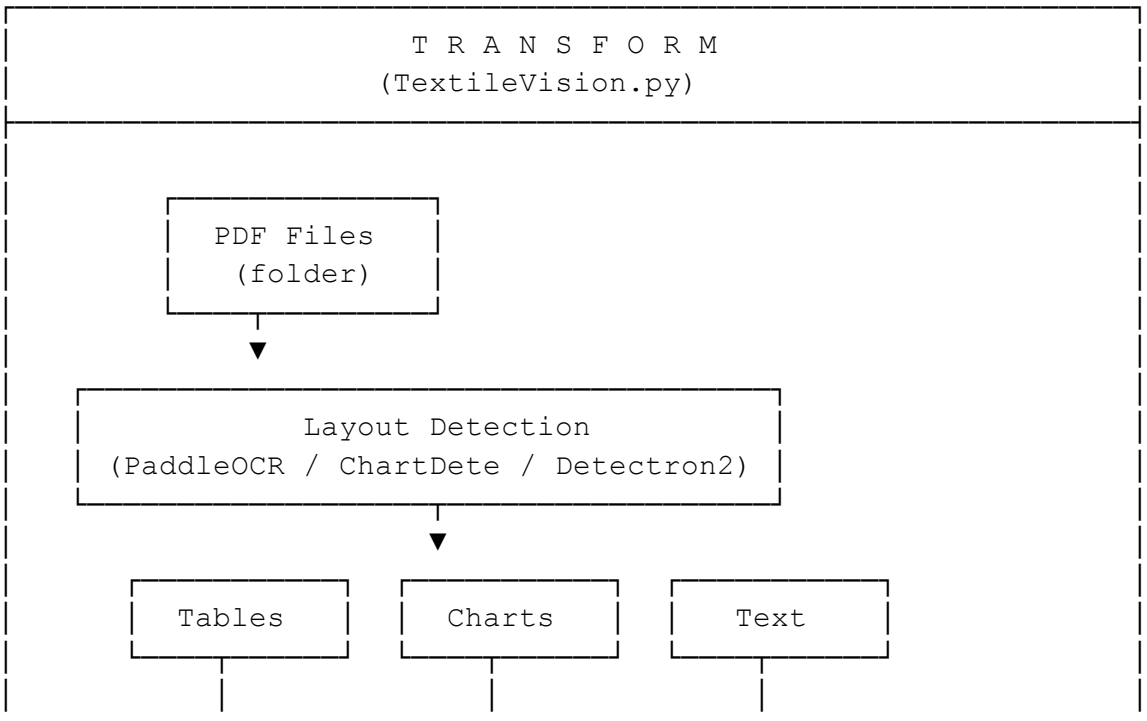
FabricETL
Export output.xlsx for analysis

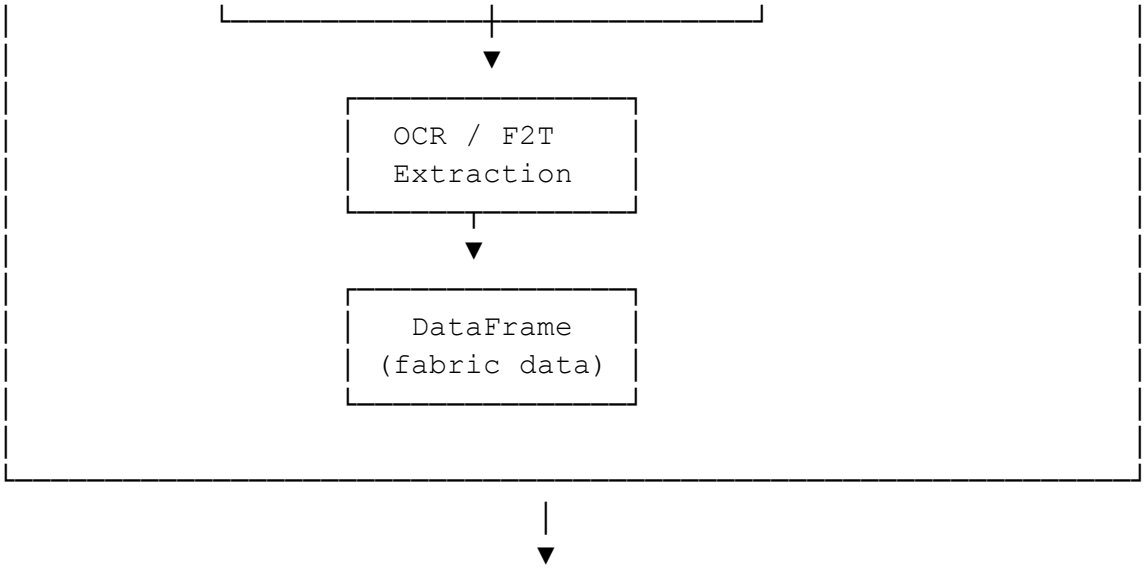
Pipeline Phases

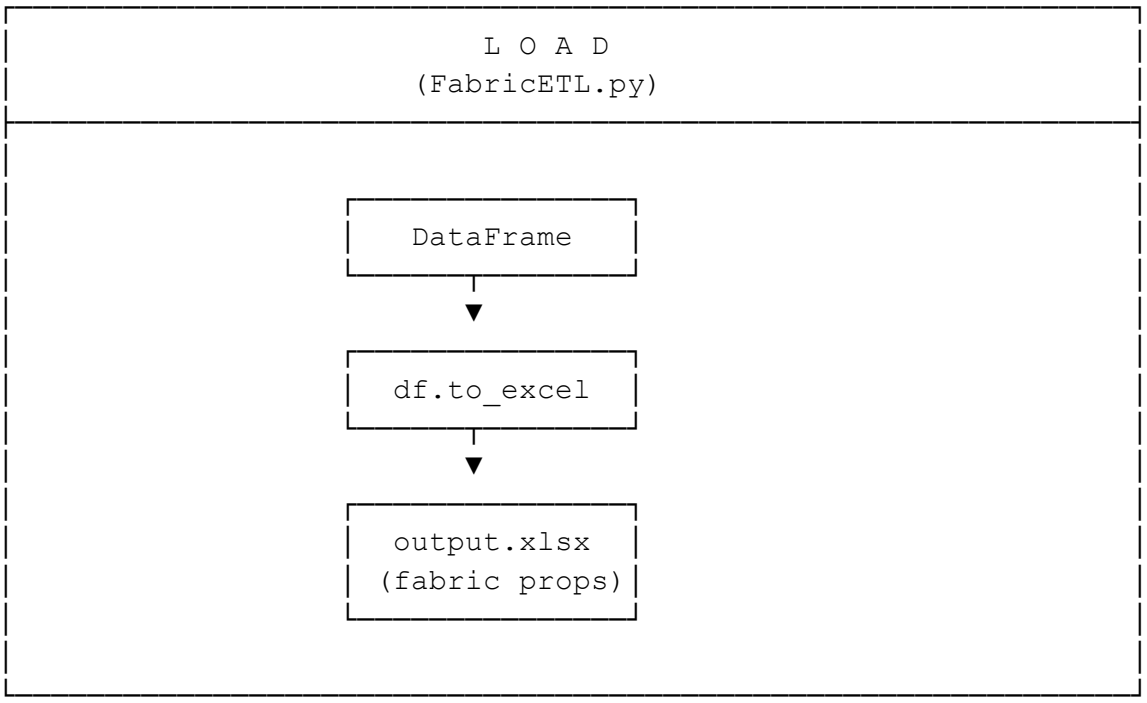
Phase	Component	From	To
E | ScholarSweep	API databases |		PDFs + metadata.xlsx
T | TextileVision	PDFs		DataFrame (structured metrics)
L | FabricETL			DataFrame | output.xlsx




OA Papers
(green)
Non-OA
(red/skip)
Download
PDFs
metadata.xlsx
(paper info)


T R A N S F O R M
(TextileVision.py)
PDF Files
(folder)
Layout Detection
(PaddleOCR / ChartDete / Detectron2)
Tables
Charts
Text





EXTRACT Phase (ScholarSweep)

1. User provides search query and parameters
2. Query APIs in parallel (CrossRef, OpenAlex, PubMed)
3. Deduplicate results by DOI
4. Check Open Access status for each paper

5. Save metadata to metadata.xlsx (green=OA, red=non-OA)
6. Download PDFs for OA papers to Downloaded Papers/ folder

TRANSFORM Phase (TextileVision)
7. Scan PDF folder for all .pdf files
8. For each PDF:
- Render pages to images
- Run layout detection (find tables, charts, figures)
- Run OCR/F2T extraction on detected regions
- Parse fabric metrics (thickness, porosity, wicking rate, etc.)
9. Combine all extracted data into single DataFrame

LOAD Phase (FabricETL)
10. Receive DataFrame from TextileVision
11. Write DataFrame to metrics.xlsx
12. Display summary (papers processed, rows extracted)

Output Files

File: metadata.xlsx
Created By: ScholarSweep
Paper titles, authors, DOIs, journals, OA status

Files: Downloaded Papers
Created By: ScholarSweep
PDF files of filtered group of papers

File: metrics.xlsx
Created By: FabricETL
Extracted fabric properties from PDFs

Many layout analysis, OCR, and Figure-to-Text packages were tested and those tested are included here as reference for those looking to build programs that need to extract text/elements. Table-LLaVa and ChartVLM-L or ChartVLM-B were finally settled on.

Tool	Layout	Text	Tables	Charts
PaddleOCR	✅		✅	✅
Detectron2 (DocLayNet)	✅			
ChartDete	✅			✅

Marker	✅	✅			
Mistral OCR		✅			
pdfTable	✅				
Camelot				✅	
pdfplumber		✅	✅		
PyMuPDF (fitz)		✅			
Tesseract		✅			
Azure Document Intelligence	✅		✅		
Google Cloud Vision	✅		✅		

5. Author Contact

This repository has GitHub's "Issues" feature enabled, so users can create what is effectively a support ticket which I receive. To do this:

1. Click the Issues tab.
2. Click New Issue.
3. Drag-and-drop text files directly into the comment box to send them to you.
4. Click Submit new issue.