Fabric Github Repository Outline
(Summary Version)

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 refines and reorganizes the dataset produced by the query. Specifically, repositories are searched for their papers, the program then performs various functions to refine the data locally, the papers are downloaded and then if the user had specified from the beginning there may be a final filtering on the downloaded papers. This results in a CSV file which represents a curated list of metadata of papers believed to be relevant to the user query subject matter.

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

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

Downloader API:

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).

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.

Prompt

```
├── CrossRef
│   ├── Title
│   ├── Abstract
│   └── Full-text

├── OpenAlex
│   ├── Title
│   ├── Abstract
│   └── Full-text
│
├── PubMed
    ├── Title
    ├── Abstract
    └── Full-text
```

Outputs: 1. The first output is a CSV 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 CSV 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 CSV 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 without paying or otherwise needing to provide 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 CSV 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.

Resume: This feature simply 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. Process Flow

Prompts allow the user to enter all query parameters for the repository searches. These parameters are then sent in API calls to the repository and written to the CSV file. Because different API's allow or disallow searching by exact phrases or searching by full-text, the process will differ depending on the requested search parameters. The user is always given the option to filter locally by the fields which are not allowed by any repository. Once the user inputs all values, the query is built and the API calls begin. Any filtering that must be done locally is done as soon as it can be done. The output CSV file is appended to and rows of papers which were not downloaded are turned red, while those that were successfully downloaded are turned green, while those which failed to download turn red. The CSV now represents a curated list of papers which are believed to be relevant to the search query topic, and the user can reference th

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.