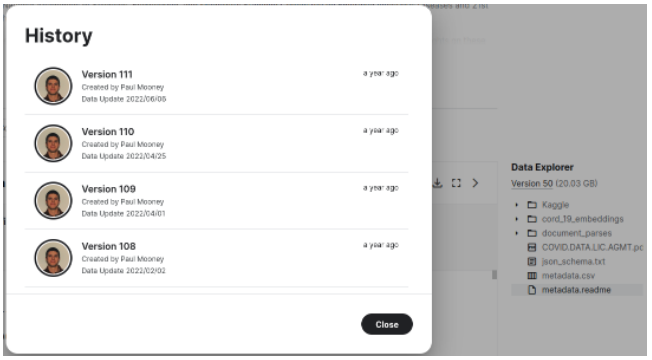
2 Analysis of Covid-19 papers

Complexity degree: ■□□

2.1 Introduction

This distributed computing project will focus on analyzing papers related to COVID-19, SARS-CoV-2, and other related coronaviruses. The dataset comprises a sub-sample extracted from the original dataset, which consists of over 1,000,000 papers and continues to grow. This dataset is part of the real-world research on COVID-19 known as the COVID-19 Open Research Dataset Challenge (CORD-19). Information about the research and associated challenges can be found on the dedicated page on Kaggle: https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge
You have the flexibility to choose the amount of data to analyze by selecting any version of the dataset you prefer. As of the time of writing, the total dataset size exceeds 80 GB. However, utilizing the Kaggle data explorer functionality, you can opt for an older dataset version (e.g., v50, or older still), as shown in the following screenshot:



2.2 Structure of the Data

All the documents present in the *data* folder are stored as JSON files with the following structure, taken from the link.
The data may contain empty fields and malformed values, so it's crucial to exercise caution during analysis and processing, and clean/filter data during pre-processing.

```
{
    "paper_id": <str>,                    # 40-character sha1 of the PDF
    "metadata": {
        "title": <str>,
        "authors": [                      # list of author dicts, in order
            {
                "first": <str>,
                "middle": <list of str>,
                "last": <str>,
                "suffix": <str>,
                "affiliation": <dict>,
                "email": <str>
            },
            ...
        ],
        "abstract": [                     # list of paragraphs in the abstract
            {
                "text": <str>,
                "cite_spans": [           # list of character indices of inline citations
```

```
                                        # e.g. citation "[7]" occurs at positions 151-154 in "text"
                                        #      linked to bibliography entry BIBREF3
                    {
                        "start": 151,
                        "end": 154,
                        "text": "[7]",
                        "ref_id": "BIBREF3"
                    },
                    ...
                ],
                "ref_spans": <list of dicts similar to cite_spans>,     # e.g. inline reference to "Table 1"
                "section": "Abstract"
            },
            ...
        ],
        "body_text": [                      # list of paragraphs in full body
                                            # paragraph dicts look the same as above
            {
                "text": <str>,
                "cite_spans": [],
                "ref_spans": [],
                "eq_spans": [],
                "section": "Introduction"
            },
            ...
            {
                ...,
                "section": "Conclusion"
            }
        ],
        "bib_entries": {
            "BIBREF0": {
                "ref_id": <str>,
                "title": <str>,
                "authors": <list of dict>       # same structure as earlier,
                                                # but without `affiliation` or `email`
                "year": <int>,
                "venue": <str>,
                "volume": <str>,
                "issn": <str>,
                "pages": <str>,
                "other_ids": {
                    "DOI": [
                        <str>
                    ]
                }
            },
            "BIBREF1": {},
            ...
            "BIBREF25": {}
        },
        "ref_entries":
            "FIGREF0": {
                "text": <str>,                  # figure caption text
                "type": "figure"
            },
            ...
            "TABREF13": {
                "text": <str>,                  # table caption text
                "type": "table"
            }
        },
        "back_matter": <list of dict>           # same structure as body_text
    }
}
```

2.3 Assignment

2.3.1 Word counter distributed algorithm

The first part of the task will see you implement a distributed algorithm to count the occurrences of all words within the list of documents. In Natural Language Processing (NLP), a document refers to a body of text, where each paper represents a document.
The algorithm is defined as follows:

Map phase : For each document D_i, produce the set of intermediate pairs $(w, cp(w))$, one for each word $w \in D_i$, where $cp(w)$ is the number of occurrences of w in D_i. E.g.: $('hello', 3)$

Reduce phase : For each word w, gather all the previous pairs $(w, cp(w))$ and return the final pair $(w, c(w))$ where $c(w)$ denotes the number of occurrences of w across all documents. In other words, $c(w)$ is equal to $\sum_{k=1}^{n} cp_k(w)$

The algorithm must be executed on the full text of all the papers considered in the dataset. To obtain the full text of a paper, you need to transform the input data by concatenating the strings contained in the `body-text` fields of the JSONs.
We recommend utilizing the RDD/Bag data structure for this transformation. However, if you prefer implementing the algorithm using the DataFrame structure, feel free to do so.
Upon completion of the algorithm, you must identify the most frequently used words, which should show their relevance to viruses and research. For instance, create a barplot of the top words, or perform any other plot to quantify your findings.

2.3.2 Which are the worst and best-represented countries/institutes in the research?

In this part of the project, you'll be required to convert the documents into DataFrame data structure to identify the countries that have been the most and least active in this research field. This analysis should be based on the country of affiliation of the authors. Do the same for the affiliation institute of the authors to identify the universities and research centers that are cited the most/the least.

2.3.3 Obtaining Embeddings for Paper Titles

In Natural Language Processing (NLP), a common technique for text analysis is to transform text into numerical vectors, where each vector represents a word within a document. At the end of the embedding phase, the document is transformed into a list of vectors or a matrix of $n \times m$ where n is the number of words in the document, and m is the size of the vector representing each word n.
For more information about word embedding, refer to this resource.
In this part of the project, you'll have to transform the title of each paper into its embedding version using a pre-trained model. There are several available versions of pre-trained English models. You can choose to use any of the ones proposed at the link, or to opt for the most complete model, available at this other link.
The pre-trained model is essentially a large dictionary in the format `key:vector`.
One of the possible ways to load the model is provided in the official FastText page, and should be re-adapted to be exploited in the context of PySpark/Dask.

Once the model is loaded, you can use the `map` approach to create a DataFrame or a Bag/RDD that is composed of:

- paper-id
- title-embedding

The title embedding can indeed be represented as either a list of vectors or flattened into a single large vector.
When represented as a list of vectors, each vector corresponds to a word in the title, forming a sequence of embeddings that captures the semantic meaning of the title.
Alternatively, when flattened into a large vector, the embeddings of all words in the title are concatenated or aggregated into a single vector representation. This approach reduces the dimensionality of the title embedding while still preserving semantic information, making it suitable for input into machine learning models that require fixed-size input vectors.
Both representations have their advantages and can be chosen based on the specific requirements of the downstream tasks or models being used.

2.3.4 Cosine Similarity Computation

Utilize the vectors generated for each paper title to compute the cosine similarity between each pair of titles.
The cosine similarity measures the similarity between two vectors by computing the cosine of the angle between them, resulting in a value between -1 and 1, where 1 indicates perfect similarity and -1 indicates "perfect dissimilarity".
You will have to compute the cosine similarity between each pair of papers' titles to identify some of the most similar (and most dissimilar) papers in the dataset.

2.4 Hints/Suggestions

- The text of the documents must be sanitized before performing the analysis (ensure that you sanitize the text of the documents *before* performing the various analyses). This typically involves removing punctuation and stop-words to focus on meaningful content.
- Keep in mind that not all words in the documents may be present in the pre-trained model. In such cases, simply skip the word and proceed to the next one in the text.
- If you're using a distributed "read file" approach, ensure that all files are located in the same path accessible to each worker. This ensures consistency and prevents errors during data processing.