1 Objectives

In this milestone, students will transition from simple data collection to advanced data processing and analysis. The focus of this Milestoneis to transform unstructured raw data (LaTeX sources) into a structured hierarchical format and apply machine learning techniques to match citation entries extracted from the text with external metadata. This practice mimics real-world data science workflows involving unstructured text processing, standardization, feature engineering, and entity resolution/matching algorithms.

2 Description

This milestone is divided into two main areas: parsing and standardizing raw LaTeX sources, and performing a reference matching task using a machine learning pipeline.

2.1 Hierarchical Parsing and Standardization

Students are required to implement an automatic parser that processes the scraped LaTeX source files from the previous Milestone. The processing pipeline consists of the following components:

2.1.1 Multi-file Gathering

The content of a LaTeX source is often spread across multiple `.tex` files (e.g., using `\input` or `\include`). Students must programmatically identify the main compilation path and parse only those files that are actually included in the final PDF rendering. Unused or redundant files found in the source directory must be ignored.

2.1.2 Hierarchy Construction

Hierarchy Structure The goal is to convert the raw LaTeX code into a hierarchical tree structure for each version of each publication.

- The **Document** acts as the root.
- **Chapters** or **Sections** typically comprise the second level.
- Other elements such as **Subsections**, **Paragraphs**, and subsequent logical divisions form the lower levels. Since the exact nesting order depends on the publication's specific formatting, the parser is required to automatically identify the appropriate hierarchical level for each such element.

Parsing Notes To ensure consistency, the parser must adhere to specific rules regarding element granularity and exclusion:

- **Smallest Elements:** The leaf nodes (smallest elements) of the hierarchy must be **Sentences** (separated by dots/periods), **Block Formulas** (mathematical blocks such as contents within `equation` environments or `$$...$$`), and **Figures** (note that **Tables** are also considered a type of Figure for this purpose). The itemized points are considered as branching into deeper levels, meaning that a block of `\begin{itemize}...\end{itemize}` is considered as a higher component with each point being a next-level element.
- **Exclusions:** References sections should **not** be parsed into the hierarchy.
- **Inclusions:** Acknowledgements and Appendices (often unnumbered using `\section*`) **must** be included.

2.1.3 Standardization and Deduplication

Cleanup and Normalization Students must clean and normalize the data within the hierarchy:

- **LaTeX Cleanup:** Remove unnecessary formatting commands (e.g., `\centering`, `[htpb]`, `\midrule`) that do not contribute to semantic meaning.
- **Math Normalization:** Convert all inline mathematics to a unified format (e.g., `$...$`) and all block mathematics to a unified format (e.g., `\begin{equation}...\end{equation}`).

Reference Extraction For citations defined using `\bibitem` within the `.tex` files, students must convert them into standard BibTeX entries using programmatic tools (e.g., Regular Expressions).

Deduplication Deduplication is required at two levels:

- **Reference Entries:** Duplicated reference entries inside each version or across different versions of a single publication should be properly handled. If different citation keys (names) refer to the same underlying reference content, a single citation key should be chosen, and the `\cite{}` commands of the remaining entries must be renamed to this chosen key. Additionally, the deduplication process should unionize the fields of the duplicate entries rather than selecting one entry while discarding all other information.
- **Full-text Content:** Full-text content deduplication must be applied to the hierarchical elements. If an element's text content matches exactly across different versions (full-text match), it should be represented by a single identifier in the final output, reducing redundancy. **Note:** Full-text matching should be performed *after* cleanup to remove discrepancies in minor details (such as formatting or spacing) that might otherwise cause the code to identify identical texts as different.

2.2 Reference Matching Pipeline

This section requires students to perform a matching task between the references extracted and standardized above and the scraped `references.json` data. The goal is to identify which extracted BibTeX entry corresponds to which arXiv ID (or metadata entry) in the `references.json` file. Students must perform the full data science pipeline as described below.

2.2.1 Data Cleaning

Perform any additional text preprocessing required to facilitate the matching process (e.g., lowercasing, stop-word removal, or specific tokenization for author names and titles).

2.2.2 Data Labelling

Students need to create a ground truth dataset by labelling associations between the extracted BibTeX entries and the entries in `references.json`. The task is to link the converted BibTeX entries from the LaTeX source to the correct arXiv ID found in the `references.json` file. Students are required to:

- Manually label references for at least **5 publications**. The manual ground truth dataset must contain a total of at least **20 labeled reference pairs** (associations between a BibTeX entry and a target arXiv ID).
- Implement automatic matching based on classic tools (such as regex, string similarity heuristics, etc.) to generate labels for at least **10%** of the remaining non-manual data.

The internal format for these labels is up to the student but must be clearly described in the final report.

2.2.3 Feature Engineering

Students must first frame the matching task as a specific supervised learning problem (e.g., classification, regression, recommendation, or ranking), which determines the input format (e.g., entry-reference pairs, triplets, or candidate lists). Based on this chosen input format, students must perform feature engineering to construct reasonable features for the model. These features can range from simple string matching scores or embeddings to complex statistical features.

Students must justify the creation of each feature in the final report, explaining the underlying idea. This insight and idea should be illustrated with appropriate data analysis.

Note: In this feature engineering phase, students are allowed to construct features based on hierarchies previously formed during the first part of this Milestone.

2.2.4 Data Modeling

The target of this problem is to match the entries inside the standardized BibTeX file with associated references inside `references.json` of the same publication.

If there are m BibTeX entries in the merged BibTeX file and n references inside the scraped SemanticScholar `references.json`, then there are $m \times n$ possible pairs inside that single publication. Thus, there are actually $m \times n$ data points generated from this publication only. Consequently, $m \times n$ labels should be assigned, where each pair has a binary label of **yes** or **no**, indicating whether the BibTeX entry from the `.bib` file corresponds to the associated reference in `references.json`.

The data split strategy is as follows:

- **Test Set:** Includes 1 publication from the manually labeled set and 1 publication from the automatically matched set.
- **Validation Set:** Includes 1 publication from the manually labeled set and 1 publication from the automatically matched set.

- **Training Set:** Consists of all remaining publications (the remaining manually labeled ones and the remaining automatically matched ones).

2.2.5 Model Evaluation

Evaluate the trained models on the **Test set** (comprising the 2 publications defined above). The model should output a **ranked list of the top 5 candidates** from `references.json` for each BibTeX entry. Post-processing of the results is allowed.

The evaluation metric is **Mean Reciprocal Rank (MRR)**.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

Where $|Q|$ is the total number of references to match, and rank_i is the rank position of the correct match within the top 5 candidates. If the correct match is not in the list, the score is **zero**.

3 Submission Format

3.1 Data Submission

Students must submit a ZIP file named after their student ID (e.g., `23456789.zip`) containing both the processed publication data and the machine learning results. This ZIP file must be uploaded to a Google Drive folder link that the lab instructor will provide later.

3.1.1 Folder Structure

Inside the main ZIP folder, there should be separate subfolders for each publication. The structure is as follows:

```
<Student ID>/
├── <yymm-id>/
│   ├── hierarchy.json
│   ├── refs.bib
│   ├── metadata.json
│   ├── references.json
│   └── pred.json
└── ...
```

3.1.2 File Descriptions

- **metadata.json**: This file stores the paper's metadata, including at minimum: the paper title (as a string), authors (as a list of strings), submission date (as a string in ISO format), and revised dates (as a list of date strings in ISO format). Additionally, include the publication venue (e.g., journal or conference name) if applicable.
- **references.json**: This file contains a dictionary where keys are arXiv IDs (in "yymm-id" format) of the crawled references, and values are their respective metadata dictionaries. Each metadata dictionary must include at minimum: the reference paper title (as a string), authors (as a list of strings), submission date

(as a string in ISO format), and Semantic Scholar ID (as provided by the Semantic Scholar API). For papers that do not have associated arXiv IDs, do not include them in this file as the IDs serve as keys.

- **refs.bib**: This file contains the unified BibTeX entries after deduplication and merging from the hierarchy construction step.
- **hierarchy.json**: This single JSON file replaces the raw `tex` folder from the previous Milestone. It contains the parsed content and the hierarchical structure. The format is defined below:

```
{
  "elements": {
    "smallest-element-id": "Cleaned latex content ...",
    "higher-component-id": "Title of the associated content"
  },
  "hierarchy": {
    "1": { // Version 1
      "higher-component-id": "root-document-id", // parent
      "smallest-element-id": "higher-component-id" // parent
    },
    "2": { // Version 2
       "id-string-x": "id-string-y" // parent
    }
  }
}
```

Listing 1: Format for hierarchy.json

Note:

1. The `elements` field contains elements from **all** versions of the publication. These elements must be deduplicated; meaning if an element's text content matches exactly across versions (full-text match), it should be represented by a single `id-string`.
2. The ID elements should contain information about the publication ID (arXiv ID or Semantic Scholar ID) to identify which publication the element belongs to, while the remaining part of the ID should discriminate between elements inside that publication.

3.1.3 Machine Learning Results

The results of the machine learning matching process must be submitted in a separate JSON file named `pred.json` located **inside** the corresponding publication subfolder.

Important Note: Only the publications that are used for the reference matching task (those in the Training, Validation, and Test sets) are required to contain this file.

```
{
  "partition": "test",
  "groundtruth": {
    "bibtex_entry_name_1": "arxiv_id_1",
    "bibtex_entry_name_2": "arxiv_id_2",
    "..."
  },
  "prediction": {
    "bibtex_entry_name_1": ["cand_1", "cand_2", "cand_3", "cand_4", "cand_5"]
  }
}
```

Listing 2: Format for pred.json

- **partition**: Indicates if the publication was used for `train`, `valid`, or `test`.
- **groundtruth**: Contains the correct mappings, where keys are the extracted BibTeX entry names (identifiers) and values are the corresponding arXiv IDs from `references.json`.
- **prediction**: Contains the model's output, where the value is a ranked list of candidate IDs corresponding to the BibTeX entry.

3.2 Source Code Submission

For source code submission, include only source files such as Python files (`.py` or `.ipynb`) in the submission to reduce file size while excluding data files, compiled executables, or any other non-source materials. In preparation for code submission, create a folder named after your student ID (e.g., "23456789") containing the following structure:

```
23456789/
|-- src # folder containing the source code
|   |-- *.{py/ipynb}
|   |-- requirements.txt # To re-produce the environment
|-- README.md # Instructions on environment setup and code execution
|-- Report.{docx/pdf} # Use your preferred format but only single file
    permitted
```

Zip the entire student ID-named folder into a single ZIP file named after your student ID (e.g., "23456789.zip") and upload it to the Moodle system for evaluation. All the submission above should be in the official language of your program.

3.2.1 Report Requirements

Students must prepare a brief report detailing the implementation process. This report should specifically cover:

- **Hierarchical Parsing:** Explanation of the logic used to detect file structure, parse the LaTeX hierarchy, and handle multi-file inputs.
- **Standardization:** Details on how data cleaning, math normalization, and reference deduplication were implemented.
- **Machine Learning Pipeline:** Justification for the selected features, description of the model used, and analysis of the evaluation results (MRR).
- **Statistics:** Relevant statistics such as the success rate of parsing and matching.

3.2.2 Demonstration Video

Students should also make a YouTube video for demonstration purpose where its link is shared through the aforementioned report.

- **Privacy:** The video must be made publicly viewable and maintained as is for at least **1 month** after the course's completion.
- **Length:** The video should be at least **240 seconds** and at most **300 seconds**.
- **Content:** Demonstrate how to run the submitted source code, show the processing steps (parsing/matching), and visualize the outputs or intermediate logs.
- **Audio:** Provide a voice-based explanation of the running process and the reasoning behind the system's design. The language can be any that the students feel comfortable with.

4 Grading criteria

Your grade is determined upon five main criteria as listed below:

- **Hierarchical Parser:** Accuracy of the parser in detecting structure, handling multi-file sources, standardization, and extracting references.
- **ML Pipeline Implementation:** Correct implementation of cleaning, labeling, feature engineering, and modeling steps for reference matching.
- **Evaluation & Analysis:** Proper use of MRR, clear justification of features, and insightful analysis of results in the report. Note that students using external-knowledge methods such as LLMs or pretrained models shall be judged in separation from the other students.
- **Code Quality & Reproducibility:** Clean, runnable code with clear instructions.
- **Report & Demo:** Clarity of the report and the demonstration video.

5 Contact

If you have any further questions for this milestone, contact the instructors at:
Huỳnh Lâm Hải Đăng: hlhdang@fit.hcmus.edu.vn
or through the previously posted ZALO group. Instructors will answer your questions as soon as possible.