Maincode Homework Task

Dr Brian Le

December 12, 2025

Contents

1. Task Description

The task given was to build an end-to-end data pipeline from a unprepared dataset. The task specifies the final dataset must be an English language dataset with the following criteria:

1. Data acquisition
2. Data cleaning, normalisation and tokenisation
3. Training ready exports (i.e. tokenised, mixtures, shards, etc.)
4. Inspectability (e.g. histograms of length, lang scores, dup markers, PII hit-rates, drop reasons)
5. Conceptual plan for scaling

The methodology and implementation of the data pipeline will be described in this report. Where applicable, decision points will be outlined along with the reasoning behind them.

2. Dataset statistics

The dataset provided consisted of 269378 rows of `jsonl` data with two columns (`text` and `url`).

From a cursory glance, the dataset consists of a mixture of data from various sources. The URL field contains 34% of null values while the text field is complete. A total of 63599 sources are represented in the dataset with the top 20 sources being:

Domain	Rows
No source	90519
`github.com`	71415
`www.taiwannews.com.tw`	3030
`en.wikipedia.org`	2326
`www.nigeriatoday.ng`	2162
`placeholder.co`	2110
`abcnews.go.com`	2060
`sample-company.net`	1997
`example.com`	1991
`testsite.org`	1964
`demo-page.info`	1938
`www.yahoo.com`	946
`uproxx.com`	844
`www.israelnationalnews.com`	746
`www.nytimes.com`	735
`www.wafb.com`	650
`newyork.cbslocal.com`	478
`www.ocregister.com`	475
`www.engadget.com`	452
`www.parrysound.com`	416

Table 1: Top 20 domains

The presence of placeholder domains such as `placeholder.co`, `example.com`, and `testsite.org` indicates that some data entries may be synthetic or used for testing purposes. This will be used as a metric to determine the efficacy of the data cleaning process. Sources such as `github.com` are likely to contain code snippets or technical documentation so may need to be treated differently during preprocessing.

Of the non-null URLs, the top 20 top level domains (TLDs) are:

Domain	Rows
`com`	128768
`org`	10330
`de`	6303
`net`	4722
`uk`	3924
`tw`	3056
`info`	2764
`co`	2503
`ng`	2203
`au`	1567
`es`	1387
`ca`	1210
`edu`	822
`at`	559
`eu`	513
`mx`	456
`ch`	444
`in`	411
`sg`	394
`ru`	380

Table 2: Top 20 top level domains

This indicates an admixture of English (`com` , `org` , `net` , `uk` , `au` , `ca` , `edu` , `eu` , `mx` , `in` , `sg`), German (`de` , `at` , `ch`), Spanish (`es`) and other languages (`tw` , `ng` , `ru`). The cleaning is likely to involve language detection and filtering to ensure the final dataset is exclusively English.

3. Data Pipeline

The dat pipleline will be divided into two stages:
1. Data Cleaning and Normalisation
2. Data Preparation for Training
3. Analysis and Visualisation of Results

For the pipeline implementation, `python3.13` will be used along with the following libraries:
- `pandas` [1]: for data ingestion, manipulation and analysis
- `langdetect` [2]: for language detection
- `lingua` [3]: for language detection
- `BeautifulSoup` [4]: for HTML tag removal
- `nltk` [5]: for tokenisation and text preprocessing
- `presidio` [6]: for PII detection and removal
- `matplotlib` [7] and `seaborn` [8]: for data visualisation and inspection

Time did not allow for experimentation of `detoxify` [9] for removal of toxic content but this would be a useful addition to the cleaning pipeline. The laptop I was working on has poor support for the `pytorch` [10] version needed for `detoxify` so I was unable to get it working in time.

Given the small size of the dataset, the entire dataset will be processed in-memory using `pandas` DataFrames. The `pipe` functionality of `pandas` will be used to chain together the various cleaning steps in a modular fashion.

Ideally this would be scaled using a distributed data processing framework such as `Apache Spark` [11], `Ray` [12] or `Dask` [13] for larger datasets.

My main focus was integrating established libraries for data cleaning and normalisation rather than building custom implementations from scratch. The next extension to this work would be benchmarking alternative libraries and then implementing a scaled version of the pipeline.

A simple `json` based configuration file will be used to specify the parameters for each cleaning step (as well as which steps to run). Details of the configuration can be found in the `README.md` file in the code repository.

3.1. Data Cleaning and Normalisation

The pipeline aims to filter and clean the dataset to meet the following criteria:
- Exclusively English language Text
- Removal of PII
- Deduplicated text
- Removal of non-alphabetic text
- Normalised text

The ordering of the cleaning steps is important to ensure minimising runtime costs. The bottleneck for this was the PII detection, the language detection and the cleaning of the text inputs. Each step which filters out rows has a flag for whether to apply the filter after calculating.

The cleaning steps are as follows:

1. `deduplicate_data`
2. `clean_text_ascii`
3. `check_text_is_alphabetic`
4. `check_text_is_hyperlink`
5. `clean_text_html`
6. `clean_short_length`
7. `extract_domain_from_col`
8. `detect_pii` (not by default)
9. `tokenise_text`

3.1.1. Deduplication

The first step in the cleaning pipeline is deduplication of the dataset based on the `text` field. Only exact duplicates are removed at this stage to reduce the dataset size for subsequent processing steps.

A native `pandas` function is used for this:

```
df.duplicated(subset=["text"])
```

Potential future work could involve fuzzy matching or semantic similarity to identify near-duplicates. This would become more computationally expensive so was not implemented in this initial pipeline.

This step needs to be run first to reduce the dataset size for subsequent processing steps and can't be batched as with other steps.

3.1.2. Normalisation

As this is a natural language dataset, usual normalisation steps of lowercasing, punctuation removal, and whitespace trimming would potentially prevent training of a robust language model. Therefore, only leading and trailing whitespace is removed from the text entries.

Further cleaning focussed on removal of HTML tags and non-printable characters.

The `BeautifulSoup` library is used for HTML tag removal as it is robust and can handle malformed HTML. Note this step should also consider XML tags if the dataset contains XML data. Given it appears to have a very low amount of XML data, this was not implemented in this initial pipeline.

For non-printable character removal, a native `python` function is used to filter out characters outside the printable ASCII range.

3.1.3. Text Filtering

Several steps are implemented to filter out unwanted text entries that are not helpful for training a language model.

1. Non-alphabetic text removal: Entries that contain a high proportion of non-alphabetic characters (e.g. numbers, symbols) are removed. A threshold of 70% alphabetic characters is used.
2. Hyperlink removal: Entries that are primarily hyperlinks or contain a high proportion of URLs are removed. A threshold of 50% URL content is used.
3. Short length removal: Entries that are too short (less than 20 characters) are removed as they are unlikely to provide meaningful context for training.
4. Removal of code snippets: Given the presence of `github.com` as a major source, code snippets may be present in the dataset

Some examples of filtered text entries:

```
2016
+1
http://daytonflyers.com/roster.aspx?rp_id=2579
+1
https://www.youtube.com/watch?v=9n1cqbvCpDg&ebc=ANyPxKrVp5Vbw0bmXfouF74NnPJYepw2qcz9QyWDyxk2
FTFJTlXfC28qai251XUg0A8XqIRJ5EMNOhyT09L1s3YaSlSpaZHHTA
;)
```

Non-alphabetic text is removed by checking whether any character in the text is an alphabetic character using the native `str.isalpha()` function. A proprotion approach (eg. $> 80\%$) is probably preferred but for brevity this was not implemented.

Several texts are predominatly composed of a single (or multiple) hyperlink(s). These are identified using a regex pattern (`re.findall(r"(?P<url>https?://[^\s]+)", text)`) to match URLs and calculating the proportion of URL content in the text.

Short length texts are removed using a simple length check on the text string. This length threshold is calculated using both the text length and the number of words (split by whitespace). The thresholds used were chosen rather arbitrarily and could be tuned further.

Finally to remove code snippets, `github.com` sourced texts are filtered out. This represents a blunt approach and could be improved by implementing code detection using regex patterns or language models trained to identify code snippets. The main concern is the other steps for cleaning may inadvertently modify code snippets which make them more difficult to use in training. Alternatively this could be simply flagged and then treated separately during training. Further work could involve identifying the programming language used with the `pygments` library (or other libraries which use an ML approach such as).

3.1.4. Language Detection

To ensure the dataset is exclusively English language text, language detection is performed using two libraries: `langdetect` [2] and `lingua` [3].

The `langdetect` library is used first as it is lightweight and provides a very simple implementation that can return multiple language matches with associated probabilities. This library can not be vectorised so the row-by-row calculation makes this a very slow step in the pipeline. As this library has little fault tolerance for noisy text, this package was used to filter out obvious non-English text for further study.

The `lingua` library is then used as an alternative option as it provides higher accuracy at the cost of increased computational resources. The `lingua` library also allows for detecting from a subset of languages and has an interface for batch processing. `lingua` also contains a quicker detection mode which sacrifices some accuracy for speed.

Another library `fast-langdetect` [14] claims to provide high speed language detection using `fasttext` models but was not explored in this initial pipeline.

A final library `langid` [15] was considered but not explored due to time constraints. The `langid` library contains a larger set of languages (97) compared to `langdetect` (55) and `lingua` (75) so may provide better coverage for edge cases. For the sake of determining only English language text, this was not explored further.

The decision was to use `lingua` for the final language detection step as it provided the best balance of accuracy and speed for this dataset.

3.1.5. PII Removal

Personally Identifiable Information (PII) removal is a step commonly performed to cleanse a dataset of sensitive information prior to training. The PII library used is the `presidio` library from Microsoft [6]. This library is widely implemented including for `Azure OpenAI` services. It utilises a mixture of named entity recognition (NER) models, regex and rule based logic to identify and redact PII from text.

This step is the most computationally expensive step in the pipeline so is performed after deduplication and language filtering to reduce the number of rows processed.

An important consideration is whether to remove rows containing PII or to redact the PII from the text. As there is a risk of losing too much data, the decision was made to redact the PII from the text rather than remove the entire row. There is a risk that if there are too many names in a given text, the context may be lost as multiple redactions result in incoherent text (eg. "John went to the store. He met with Mary and they talked about Peter." becomes "`<PERSON>` went to the store. He met with `<PERSON>` and they talked about `<PERSON>`."). Note this step is not enabled by default due to the computational cost.

3.2. Data Preparation for Training

To prepare for training, the dataset is shuffled, split and tokenised.

3.2.1. Shuffling and Splitting

Shuffling is performed using the native `pandas` function:

```
df.sample(frac=1, random_state=42)
```

A splitting function is implemented using the `scikit-learn` [16] to divide the dataset into training, validation and test sets. This is a very simple split that is quite flexible. If maximising data is required, k-fold cross validation could be implemented instead. The splitting ratios are configurable via the `json` configuration file. A random seed can be set for reproducibility.

If a specific distribution of classes is required, a stratification using a particlular column can be configured. This will ensure the sampled sets maintain the same class distribution as the original dataset. One example would be to stratify on the source domain to ensure all sources are well represented in each split.

3.2.2. Tokenisation

Several libraries provide tokenisation functionality including `nltk` [5], `spacy` [17] and `tiktoken` [18]. All three are implemented and can be used to do a token count if desired.

The `nltk` and `spacy` libraries provide general purpose tokenisation suitable for natural language text whereas `tiktoken` is specifically designed for tokenising text for `OpenAI` models.

4. Results

After running the cleaning pipeline with the default settings (deduplication, normalisation, text filtering, language detection), the final dataset statistics are as follows:

Metric	Value
Initial Rows	269378
Final Rows	152974
Rows Removed	116404
Percentage Retained	56.8%

Table 3: Final dataset statistics

The final dataset consists of 152974 rows of English language text. This is quite a significant reduction from the initial dataset size of 269378 rows, indicating the cleaning steps were effective in filtering out unwanted data. Some work would be required to determine whether any data could be recovered by loosening some of the filtering criteria.

4.1. Analysis and Visualisation

To inspect the results of the cleaning pipeline, several visualisations were created using `matplotlib` [7] and `seaborn` [8].

The final top 20 domains are as follows:

Metric	Value
Initial Rows	269378
Final Rows	152974
Rows Removed	116404
Percentage Retained	56.8%

Table 4: Top 20 domains after cleaning

From the results we see no further presence of placeholder domains. The remaining top 20 TLDs are:

Domain	Rows
`com`	47267
`org`	7506
`uk`	3867
`tw`	2957
`net`	2016

Domain	Rows
`ng`	1766
`au`	1512
`ca`	1019
`edu`	768
`info`	564
`in`	401
`ru`	316
`us`	289
`sg`	268
`za`	261
`nz`	251
`eu`	242
`co`	231
`de`	221
`gov`	214

Table 5: Top 20 top level domains after cleaning

Interestingly enough, the `de` TLD remains in the top 20 indicating some German language text has passed through the language filtering step. Russian (`ru`) and Indian (`in`) TLDs are also present indicating some non-English text remains.

Remaining `tw` TLD sourced texts (mostly from `www.taiwannews.com.tw`) are English language news articles so have passed the language filtering step.

Further inspection of some `ru` TLD sourced texts (mostly from `http://www.ntv.ru`) shows they are Russian language news articles but structured in `Javascript` code. These texts need to be manually excluded.

4.1.1. Language Detection Results

A bar chart showing the distribution of detected languages before cleaning was created to visualise the effectiveness of the language filtering step. About 89% of the subsampled 10k rows are detected as English by `lingua` with smaller a small German (5.9%) and Spanish (4.9%) presence.

Given the small % of non-English text, it may be more beneficial to loosen the language filtering criteria to retain more data (`lingua` has access to a fast build).

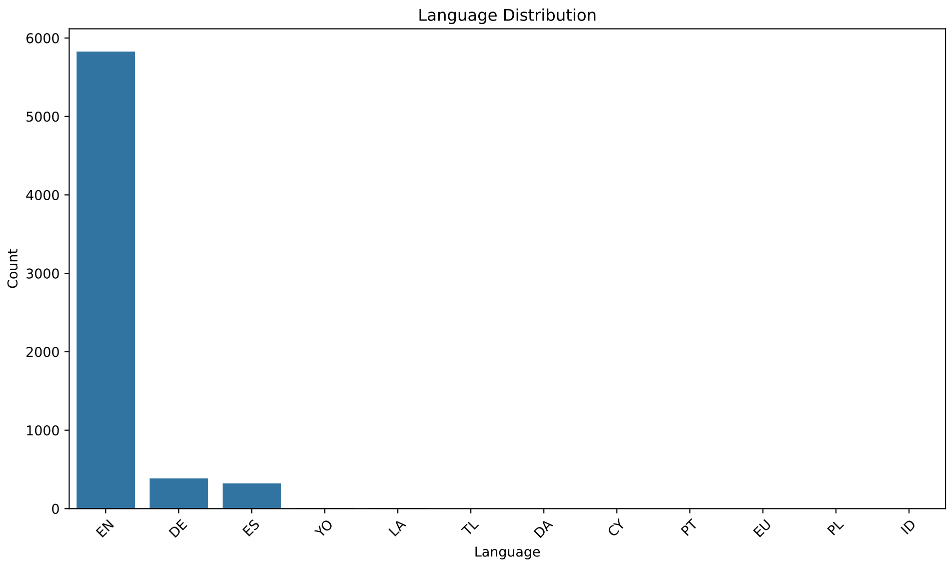

Figure 1: Language Detection Before Cleaning (subsample of 10k rows)

4.1.2. Text Length Distribution

A histogram of text lengths (in characters) after cleaning was created to visualise the impact of the cleaning steps on text length. The cleaning steps have effectively removed very short texts (less than 50 characters or 20 words). The typical length of texts appears to be ~1000 characters with 90% being below 3000 characters.

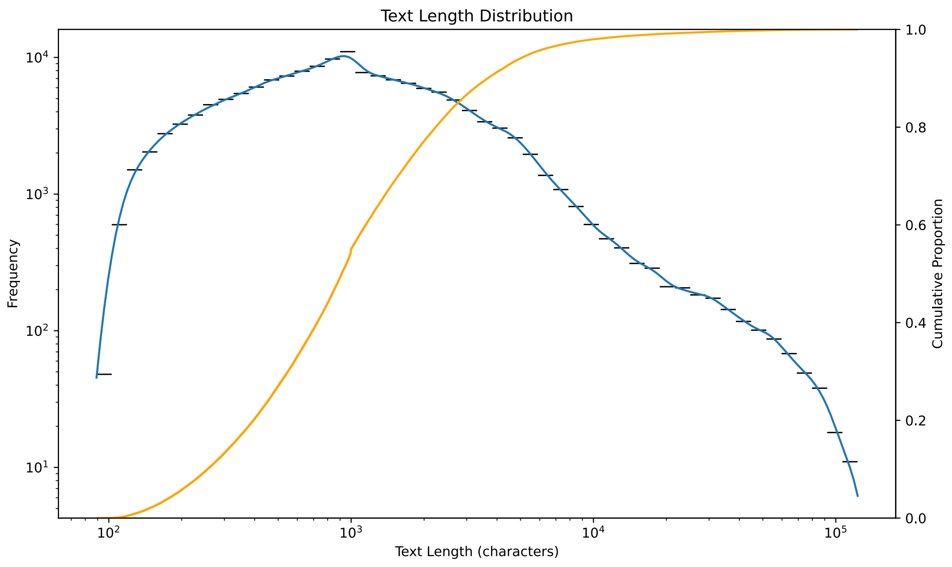

Figure 2: Text Length Distribution After Cleaning

4.1.3. Word Count Distribution

Similarly, a histogram of word counts after cleaning was created. The cleaning steps have effectively removed very short texts (less than 20 words). The typical length of texts appears to be ~200 words with 90% being below 500 words.

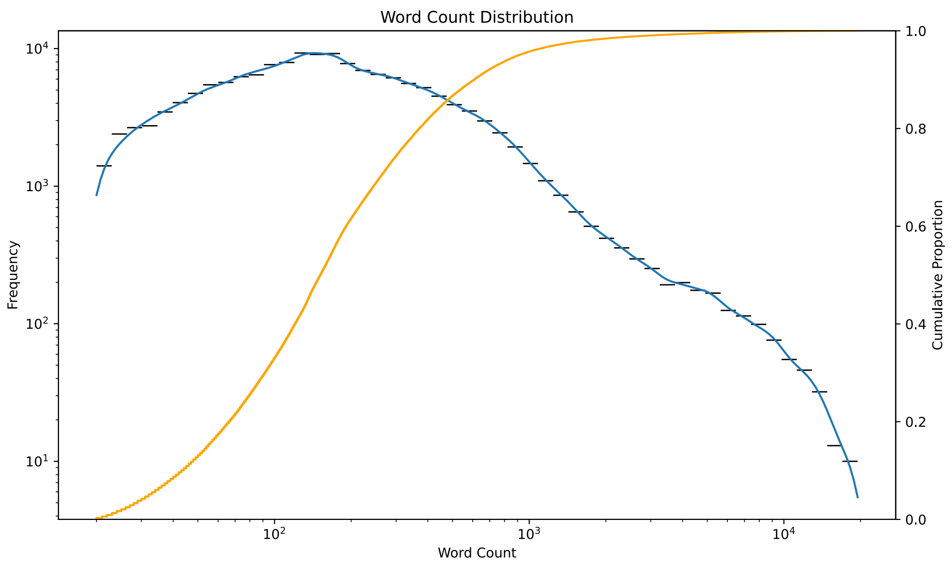

Figure 3: Word Count Distribution After Cleaning

4.1.4. Token Count Distribution

A histogram of token counts after cleaning was created using the `nltk` tokenizer. The typical length of texts appears to be ~250 tokens with 90% being below ~1000 tokens.

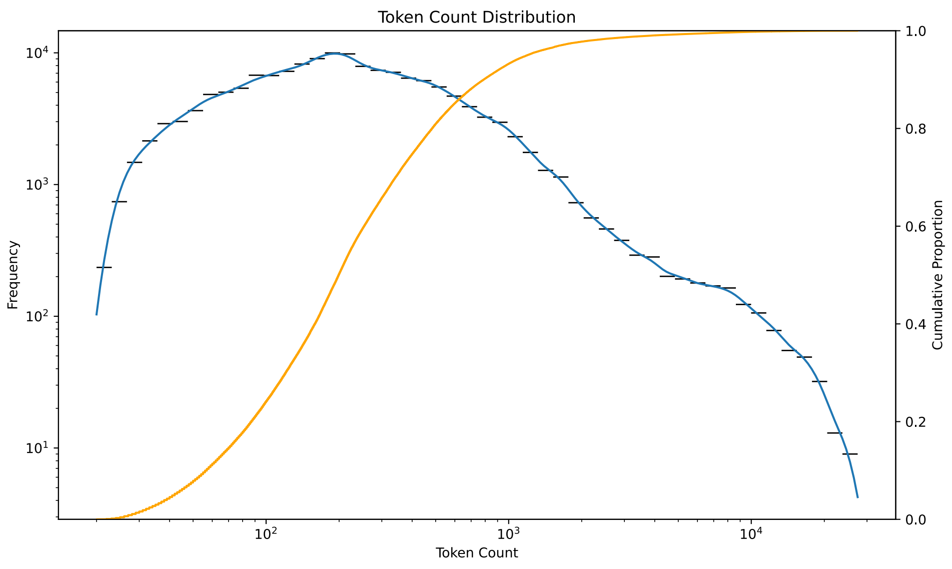

Figure 4: Token Count Distribution After Cleaning (nltk tokenizer)

4.1.5. PII Hit Rates

For completeness, a single run was performed with PII detection enabled to assess the hit rates of various PII types in the dataset.

The result was 127659 of 152974 rows (83.5%) contained at least one PII entity. This is far too loose as the majority of the dataset would be removed if rows with PII were to be excluded.

In future perhaps a subset of PII types could be used to filter rows (eg. only names and addresses).

5. Future work

The primary concern for this pipeline is scaling to larger datasets. The current implementation processes the entire dataset in-memory using `pandas` DataFrames.

For larger datasets some batch processing or distributed data processing framework would be needed. Options include:

- `Apache Spark` [11]: widely used distributed data processing framework with support for Python via `PySpark`.
- `Dask` [13]: parallel computing library that integrates well with `pandas` and can scale from a single machine to a cluster.
- `Ray` [12]: distributed computing framework that can be used for parallel data processing
- `Prefect` [19] or `Airflow` [20]: workflow orchestration tools that can manage complex data pipelines and integrate with distributed processing frameworks

Depending on the size of the dataset, a distributed processing framework may be necessary to handle the computational load. For smaller datasets (GB scale), batch processing using `pandas` or `Dask` may be sufficient. Otherwise, a full distributed processing framework such as `Apache Spark` or `Ray` may be required.

Of note the particular steps which need efficient batching are language detection and PII removal. To replace either step, other packages would need to be benchmarked for accuracy and speed.

The `detoxify` library for toxic content removal would also be a useful addition to the cleaning pipeline. Some work is required to get this working with my particular setup but should in principle be straightforward to implement.

Bibliography

[1] pandas development team, “pandas: Python Data Analysis Library.” 2023.

[2] S. Nakatani and contributors, “langdetect: Language detection library ported from Google's language-detection.” 2010.

[3] O. / lingua contributors, “Lingua: Language detection library.” 2020.

[4] L. Richardson, “Beautiful Soup 4.” 2023.

[5] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

[6] Microsoft Presidio contributors, “Microsoft Presidio: PII detection and anonymization.” 2021.

[7] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[8] M. Waskom and contributors, “seaborn: statistical data visualization.” 2021.

[9] U. A. contributors, “detoxify: Detoxification models for text (toxicity detection).” 2020.

[10] A. Paszke, S. Gross, and others, “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” 2019.

[11] M. Zaharia and others, “Apache Spark: A Unified Engine for Big Data Processing,” in *Communications of the ACM*, 2016.

[12] R. Nishihara and contributors, “Ray: A Distributed Framework for Emerging AI Applications.” 2019.

[13] M. Rocklin and contributors, “Dask: Parallel computation with task scheduling.” 2015.

[14] LlmKira, “fast-langdetect: Fast language detection using fastText models.” 2023.

[15] A. K. K. Lui and T. Baldwin, “langid.py: An Off-the-shelf Language Identification Tool,” *Proceedings of the ACL 2012 System Demonstrations*, pp. 25–30, 2012.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, and others, “scikit-learn: Machine Learning in Python.” 2011.

[17] E. AI and contributors, “spaCy: Industrial-strength Natural Language Processing in Python.” 2024.

[18] OpenAI, “tiktoken: Tokeniser for OpenAI models.” 2023.

[19] I. Prefect Technologies, “Prefect: Dataflow automation for humans.” 2020.

[20] A. S. Foundation, “Apache Airflow: A platform to programmatically author, schedule and monitor workflows.” 2015.

Index of Figures

Index of Tables