

# TABLIB: A DATASET OF 627M TABLES WITH CONTEXT

A PREPRINT

Gus Eggert, Kevin Huo, Mike Biven, and Justin Waugh  
Approximate Labs\*, Boulder, CO, USA

October 13, 2023

## ABSTRACT

It is well-established that large, diverse datasets play a pivotal role in the performance of modern AI systems for text and image modalities. However, there are no datasets for tabular data of comparable size and diversity to those available for text and images. Thus we present "TabLib", a compilation of 627 million tables totaling 69 TiB, along with 867B tokens of context. TabLib was extracted from numerous file formats, including CSV, HTML, SQLite, PDF, Excel, and others, sourced from GitHub and Common Crawl. The size and diversity of TabLib offer considerable promise in the table modality, reminiscent of the original promise of foundational datasets for text and images, such as The Pile and LAION.

## 1 Introduction

The importance of data in model training has continued to grow [Hoffmann et al., 2022]. Training data volume is now considered to be roughly as important to model performance as model size [Zha et al., 2023a]. This implies that large datasets are promising assets for improving the performance of AI models.

For example, in 2021 OpenAI released both CLIP and DALL-E [Radford et al., 2021, Ramesh et al., 2021], which were considered state-of-the-art for image tasks. A large part of their success was due to their training data scale of 400M image-text pairs, whereas previously the largest open dataset for image-text pairs was around 10M [Schuhmann et al., 2021]. Even larger training datasets such as LAION-5B [Schuhmann et al., 2022] have fueled subsequent image models like Stable Diffusion [Rombach et al., 2022].

Given the volume and significance of information captured in tabular data, research on applying AI models to tabular data is an area of active research [Badaro et al., 2023, Jin et al., 2022, Dong et al., 2022]. Despite this, there are not many large-scale, diverse, and accessible datasets for tabular data. We are aware of only one large scale crawl that exceeds 10M tables (WebTables [Lehmberg et al., 2016]), and only a few additional datasets have more than one million tables (WikiTables [Bhagavatula et al., 2015], GitTables [Hulsebos et al., 2023], VizNet [Hu et al., 2019]). Furthermore, the largest of these datasets (WebTables) is composed solely of HTML tables, which differ meaningfully from other common table types such as database tables, suggesting that WebTables may be insufficient for training models for diverse tasks. We believe that a larger and more diverse dataset will accelerate the advancement of tabular AI systems.

Thus, we present "TabLib", whose notable characteristics include:

- **Scale:** Over 627 million individual tables totaling 69 TiB
- **Table metadata:** 867B tokens of contextual information, such as filenames, URLs, text before and after the table in the source document, and OpenGraph metadata.
- **Diversity:** Across language, category, size, source (Common Crawl<sup>1</sup> and GitHub<sup>2</sup>), and format (CSV, HTML, PDF, Excel, SQLite, etc.)

\*research@approximatelabs.com

<sup>1</sup><https://commoncrawl.org>

<sup>2</sup><https://github.com>

- **Provenance:** Table source and transformation data to enable attribution and validation

These characteristics suggest TabLib could be a useful research asset for many fields, which we discuss later in [1.2 Impact](#). We hope that TabLib will help advance tabular data understanding and catalyze the development of AI models focused on this modality, which we refer to as *large data models*.

## 1.1 Related Work

Numerous open datasets exist for the purpose of training machine learning models to understand and interpret tabular data. Some of the most significant of these datasets are detailed in Table 2 in [\[Badaro et al., 2023\]](#). While high quality, existing datasets such as Spider, WikiDB, and VizNet [\[Vogel and Binnig, 2023\]](#), [\[Yu et al., 2019\]](#), [\[Hu et al., 2019\]](#) lack the size and/or diversity necessary to pre-train large data models with broad applicability.

Two data sets have noteworthy volume: WebTables [\[Cafarella et al., 2008\]](#) and GitTables [\[Hulsebos et al., 2023\]](#).

The latest WebTables corpus contains 233 million tables extracted from HTML pages from Common Crawl<sup>4</sup>. WebTables contains a large volume of tables, but has limited diversity due to only including HTML tables from web pages.

GitTables is a continuously updated library of tables extracted from “comma-separated value” files (CSVs) hosted on GitHub, containing 1 million tables. These tables tend to be structurally different from the HTML-centric WebTables [\[Hulsebos et al., 2023\]](#), thus an important table corpus. Compared to WebTables, GitTables is relatively small, and still only supports a single file type (CSV).

## 1.2 Impact

Applying AI to tabular data is an active field of study, and there are many applications and research areas that could significantly benefit from a large, diverse dataset such as TabLib. These include:

- **Dataset Search:** Identifying corresponding tables using a set of keywords that describe the required information [\[Benjelloun et al., 2020\]](#), [\[Chapman et al., 2020\]](#), [\[Zhang and Balog, 2018\]](#)
- **Semantic Understanding:** Using data tables to create or augment general-purpose knowledge bases, and vice versa. [\[Dong et al., 2014\]](#), [\[Liu et al., 2023\]](#), [\[Jiménez-Ruiz et al., 2020\]](#), [\[Efthymiou et al., 2017\]](#), [\[Bonfitto, 2021\]](#), [\[Hulsebos et al., 2019\]](#)
- **Data Integration:** Identifying tables that can be joined or unioned within a large corpus of tables. Includes schema mapping. [\[Dong et al., 2021\]](#), [\[Zhang and Balog, 2019\]](#), [\[Zhu et al., 2019\]](#), [\[Nargesian et al., 2018\]](#), [\[Santos et al., 2021\]](#), [\[Srinivas et al., 2023\]](#), [\[Zhu et al., 2017\]](#), [\[Cong et al., 2023a\]](#), [\[b\]](#)
- **Knowledge Extraction:** Interacting with data through natural language, via tasks like question answering and semantic parsing. [\[Zha et al., 2023b\]](#), [\[Cheng et al., 2023\]](#), [\[Zhang et al., 2023\]](#), [\[Li et al., 2023\]](#), [\[Pourreza and Rafiei, 2023\]](#), [\[Talmor et al., 2021\]](#), [\[Lin et al., 2020\]](#)
- **Table Metadata Prediction:** Predicting metadata such as column types, inclusion of personally identifiable information (PII), and data cleanliness. [\[Zhang, 2017\]](#), [\[Parikh et al., 2020\]](#), [\[Korini and Bizer, 2023\]](#)
- **Table Representation Learning:** Representing tables as a distinct modality of information for training machine learning models [\[Yin et al., 2020\]](#), [\[Deng et al., 2020\]](#), [\[Tang et al., 2021\]](#), [\[Herzig et al., 2020\]](#), [\[Iida et al., 2021\]](#)

## 2 Methods

### 2.1 System Architecture

We built a processing pipeline that consumes raw data from data sources, extracts tables into Pandas dataframes [\[McKinney, 2010\]](#), serializes those dataframes into Arrow tables<sup>5</sup>, stores each in blob storage and metadata in a SQL database, and then aggregates into Parquet files<sup>6</sup>. To orchestrate this process, we used the Ray distributed processing framework [\[Moritz et al., 2018\]](#).

Because parsing tabular data is relatively complex compared to text due to its additional structure and data types (see [Formats, Parsing, and Metadata](#)), we encountered some failure scenarios which were difficult to recover gracefully from,

<sup>4</sup><https://webdatacommons.org/webtables/#results-2015>

<sup>5</sup><https://arrow.apache.org/>

<sup>6</sup><https://parquet.apache.org/>

such as out-of-memory errors and catastrophic regular expression backtracking. As such, we isolated each “source” as its own task instead of batching them together.

This granular task scheduling necessitated scheduling hundreds of millions of tasks. We found Ray’s scheduler problematic for this, so we scheduled these tasks using a PostgreSQL database, and used Ray to maintain long-running tasks which pulled work from the database, extracted the tables and metadata, stored the tables in blob storage, and wrote the metadata back to the DB. A separate Ray actor tracked the progress of these tasks, handled timeouts and retries, occasionally aggregated batches of metadata into Parquet files, and wrote those into blob storage.

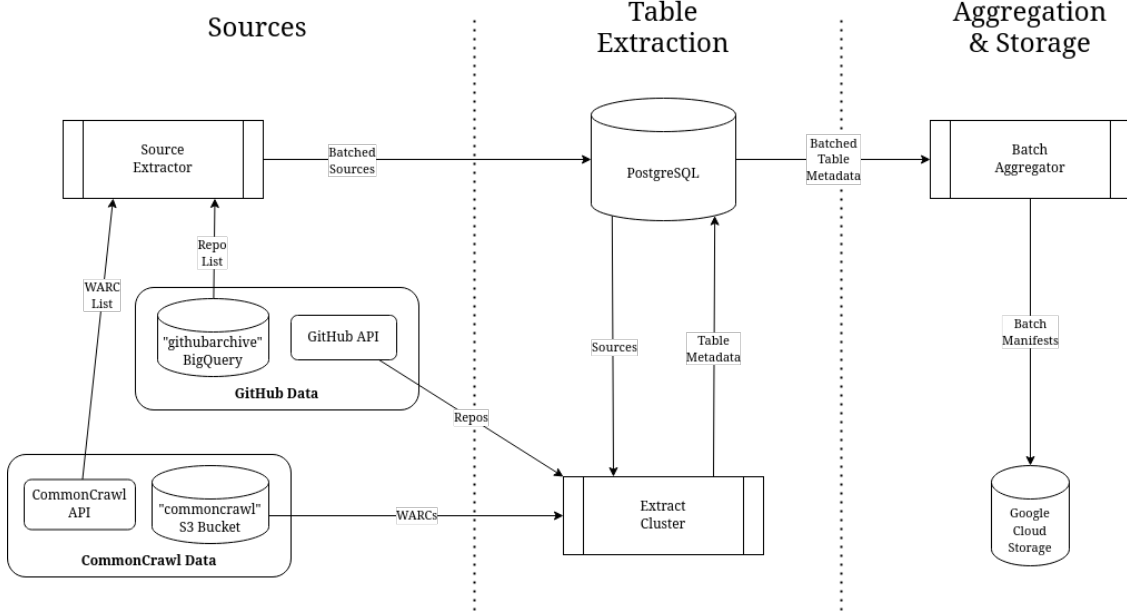


Figure 1: Architecture of table extraction pipeline.

## 2.2 Sources

For data about the number of tables extracted for each data source and file types, see [Summary Statistics](#). For samples of extracted tables and metadata, see [Sample Data](#) in the appendix.

### 2.2.1 GitHub

To reduce the amount of noise, we skipped all files under `node_modules` directories, and all JSON and YAML files which are generally configuration files in GitHub. Since files in GitHub often contain extensions like `.csv` that provide hints for the content type, we used Python’s `mimetypes.guess_type()` function to see if the file was a supported type; if not then we inspected the file’s bytes using `libmagic`<sup>7</sup>, and if it was still unsupported then the file was skipped. Files larger than 1 GB were also skipped.

Tables extracted from GitHub repos result in the following fields in each table’s `context_metadata`:

- **github\_repo:** the repo name
- **github\_ref:** the ref used, such as “refs/heads/master”
- **github\_hash:** the shortened Git commit hash
- **github\_repo\_path:** the path of the file in the repo where the table was found

### 2.2.2 Common Crawl

We used the latest crawl at the time, which was CC-MAIN-2023-23. Common Crawl results are serialized using the WARC format, which includes “request” and “response” records. We only considered response records. We discarded

<sup>7</sup><https://www.darwinsys.com/file/>

“truncated” responses which had response lengths that exceed Common Crawl’s limit. If a WARC-Identified-Payload-Type record header was included in the record, then we used its mimetype as a hint for detecting the content type, otherwise we used the Content-Type header in the HTTP response, and followed a similar approach as GitHub (use the mimetype if possible, otherwise use libmagic). About 20% of WARC files were dropped due to issues parsing certain HTML elements with Pandas.

Tables extracted from Common Crawl WARC records result in the following fields in each table’s **context\_metadata**:

- **warc\_path**: the path of the WARC file in Common Crawl
- **warc\_record\_id**: the record ID in the WARC file as specified by WARC-Record-ID
- **warc\_target\_uri**: the target URI of the HTTP request as specified by WARC-Target-URI
- **warc\_date**: the date of the request as specified by WARC-Date

## 2.3 Storage Data Model

In order to efficiently store and manage the large volume of tabular data in TabLib, we implemented a data storage model that consists of two main components: blob storage and manifests. Using this storage model, we can efficiently manage and retrieve the tables based on their metadata and content hash. This allows for easy deduplication, querying, and analysis of the dataset. A final post-processing step was performed which added the serialized tables as a column in the manifests, which is ultimately the TabLib schema, but this paper will focus on the intermediate representation because it is what the analyses are based on.

### 2.3.1 Manifest Schema

The manifests contain metadata about the tables and are stored as partitioned Parquet files. The schema for the manifests includes the following fields:

- **bucket**: the blob storage bucket of the table
- **key**: the blob storage key of the table
- **ref**: a human-readable string describing how the table was extracted
- **ref\_id**: a base64-encoded sha256 hash of the ref
- **exec\_id**: a UUIDv7 generated at the time of table extraction
- **run\_metadata**: serialized JSON object containing metadata about the run, including start and end times
- **context\_metadata**: serialized JSON object containing metadata about the table, including:
  - **extractor**: the extractor used for this table (e.g. “html”, “csv”, “pdf”, etc.)
  - **mime\_type**: the detected mime type of the bytes that the table was extracted from, e.g. “text/html” for an HTML page, “text/csv” for a CSV file, etc.
  - **<source-specific>**: additional fields depending on the source, see [Sources](#)
  - **<datatype-specific>**: additional fields depending on the data type, see [Formats, Parsing, and Metadata](#)

### 2.3.2 Blob Storage Key Schema

Each table in its intermediate form, before the final post-processing step, is stored as a separate blob object. The blob’s content is computed by serializing the Arrow table to bytes, and compressing these bytes with gzip. Each table is assigned a unique key based on the arrow table bytes content hash. The blob storage follows the following key schema:

- /manifests/{batch}/manifest.parquet
- /tables/{batch}/{base64\_sha256\_of\_arrow\_table}

## 2.4 Formats, Parsing, and Metadata

Parsing tabular data presents unique challenges that are not present when parsing text. Tasks such as inferring column data types and row delimiters are complex and error-prone. Because of this, we reused existing open-source parsers as much as possible, such as those in Pandas and pdfplumber. For most file types, we drop parsed tables with only one column, one row, all empty column names, or only numeric column names.

Below we detail each data type and a summary of the parsing logic:

Data Type	Method	Context Metadata
HTML	Parse with BeautifulSoup using lxml and html5lib parsers. Then extract all <table> elements with <code>pandas.read_html()</code> . Extract HTML metadata with <code>metadata_parser</code> library. Extract “before” and “after” context with BeautifulSoup. Drop <table> elements with colspan values > 1000 to avoid causing out-of-memory errors.	<ul style="list-style-type: none"> <li>• html_title</li> <li>• html_metadata</li> <li>• before</li> <li>• after</li> </ul>
PDF	Use pdfplumber to extract tables. Only supports text-based tables and not image-based tables, and multi-page tables appear as a separate table per page.	<ul style="list-style-type: none"> <li>• pdf_bbox</li> <li>• pdf_page</li> <li>• pdf_metadata</li> <li>• before</li> <li>• after</li> </ul>
SQLite	Use a custom SQLite VFS implementation to load the in-memory bytes with apsw, list the tables, and then parse each table with <code>pandas.read_sql()</code> .	<ul style="list-style-type: none"> <li>• sqlite_table</li> <li>• sqlite_other_tables</li> </ul>
Excel	There are many Excel formats, mimetypes, and extensions, so ignore specifics and always try parsing as XLSX using <code>openpyxl</code> , and then fall back to XLS using <code>xlrd</code> , using <code>pandas.read_excel()</code> . Parse each sheet as its own table.	n/a
Parquet	<code>pandas.read_parquet()</code>	n/a
JSON	<code>pandas.read_json(orient="records")</code>	n/a
YAML	Use <code>yaml.safe_load()</code> to convert to JSON, then <code>pandas.read_json(orient="records")</code> .	n/a
CSV	<code>pandas.read_csv(engine="python")</code>	n/a
TSV	<code>pandas.read_csv(engine="python")</code>	n/a

Table 1: **Summary of supported data types**, and how each was parsed.

### 3 Analysis and Results

#### 3.1 Keys and Metadata

We begin by examining the cardinalities of different keys: `exec_id`, `ref_id`, `key`, and `content_hash`, as shown in [Table 2](#). Definitions of these values are in [Manifest Schema](#) and [Blob Storage Key Schema](#).

The `exec_id` is unique across the dataset, generated upon line-item creation in the manifest. Any duplication indicates a serialization error.

The `ref_id` represents a unique source for a table. This should be unique across TabLib, but the current version of TabLib has some repetitions due to a bug in deduping items in the work queue. Future versions will allow tracking external data changes over time via `ref_id`.

The number of unique key values is substantial but not as large as unique `ref_id` values. This discrepancy arises because the same content table can appear multiple times within a batch (e.g., a CSV file stored multiple times in a GitHub repository with different filenames). However, `key` is not a global content-collision key as it includes the batch.

Key Type	Number of Tables
<code>exec_id</code>	660840556
<b><code>ref_id</code></b>	627208299
<code>key</code>	459022959
<code>content_hash</code>	201376283

Table 2: **Unique counts of key-like values**, ordered by decreasing uniqueness. Each may be considered some definition of “table”. We will use `ref_id` as the definition of “table” for our analyses.

Table `content_hashes` are 30.5% the size of `exec_id` values, indicating that most tables are not globally unique by content. The breakdown of these repeated tables is discussed further in [Data Duplication](#).

For clarity, the term *table* henceforth refers to a specific `ref_id` instance.

### 3.2 Summary Statistics

We calculate the total number of tables, total uncompressed table bytes, and total columns, broken out by data source and file type. See [Table 3](#) for a summary of the dataset statistics.

Source	File Type	Tables	Bytes	Columns	Metadata Tokens		
					Ref	Column Names	Context Metadata
Common Crawl	CSV	90,667	3.10 GiB	2,265,499	15,630,941	12,488,110	17,984,442
	Excel	143,012	3.26 GiB	1,836,491	21,063,876	10,579,484	60,755,137
	HTML	219,397,657	702.95 GiB	1,076,171,440	29,602,279,431	3,686,722,801	493,085,023,697
	JSON	70,737	1.75 GiB	537,934	2,737,873	4,826,400	3,393,257
	Parquet	1	4.63 MiB	13	123	30	161
	PDF	11,442,231	30.48 GiB	46,514,046	1,876,490,940	432,038,521	18,927,559,013
	SQLite	1,408	83.70 MiB	8,839	186,687	17,973	329,783
	TSV	4,374	419.82 MiB	75,989	569,475	210,506	696,084
	YAML	3,185	67.58 MiB	2,236	31,849	4,601	37,514
	Total	231,153,272	742.10 GiB	1,127,412,487	31,518,991,195	4,146,888,426	512,095,779,088
GitHub	CSV	122,091,982	59.86 TiB	5,481,784,256	7,390,202,751	36,457,207,467	13,912,319,966
	Excel	15,787,659	3.02 TiB	243,597,019	951,834,206	2,016,629,675	5,775,869,104
	HTML	199,059,080	630.31 GiB	959,028,450	11,817,543,971	2,515,057,895	173,115,916,693
	PDF	40,022,516	79.00 GiB	144,006,906	3,344,243,232	854,802,039	51,385,307,211
	SQLite	14,919,675	3.52 TiB	84,554,112	728,970,698	165,104,490	7,405,534,796
	TSV	4,174,115	1.54 TiB	94,845,931	256,836,169	739,989,036	494,089,540
	Total	396,055,027	68.62 TiB	7,007,816,674	24,489,631,027	42,748,790,602	252,089,037,310
Total	CSV	122,182,649	59.86 TiB	5,484,049,755	7,405,833,692	36,469,695,577	13,930,304,408
	Excel	15,930,671	3.02 TiB	245,433,510	972,898,082	2,027,209,159	5,836,624,241
	HTML	418,456,737	1.30 TiB	2,035,199,890	41,419,823,402	6,201,780,696	666,200,940,390
	JSON	70,737	1.75 GiB	537,934	2,737,873	4,826,400	3,393,257
	Parquet	1	4.63 MiB	13	123	30	161
	PDF	51,464,747	109.48 GiB	190,520,952	5,220,734,172	1,286,840,560	70,312,866,224
	SQLite	14,921,083	3.52 TiB	84,562,951	729,157,385	165,122,463	7,405,864,579
	TSV	4,178,489	1.54 TiB	94,921,920	257,405,644	740,199,542	494,785,624
	YAML	3,185	67.58 MiB	2,236	31,849	4,601	37,514
	Total	627,208,299	69.35 TiB	8,135,229,161	56,008,622,222	46,895,679,028	764,184,816,398

Table 3: **Summary statistics table**, showing counts of tables, bytes, columns, and tokens across GitHub and Common Crawl and the encountered file types.

We also consider token counts from metadata fields. We used `tiktoken`<sup>8</sup> to tokenize the `ref`, space-separated column names, and `context_metadata`. Because `context_metadata` has nested JSON, we considered tokenizing the string of recursively-concatenated string values, instead of the serialized JSON itself (which includes JSON syntax such as commas, curly braces, and quotation marks). We compared this on a sample and found ~10% less token counts in the JSON vs non-JSON versions. We decided that was tolerable, so we treated `context_metadata` as serialized JSON.

### 3.3 Power-Law Like Distributions

In examining TabLib, we found several metrics—including row-count, column-count, and domain-size (column-level unique-count) displaying distributions resembling power-law or Zipfian distributions, common in natural and social phenomena [Newman, 2005]. Such distributions in our data suggest a few tables or columns hold most data, while the majority hold little. This pattern can significantly impact the design and evaluation of machine learning algorithms.

Power-law distributions are characterized by an exponent or Zipf’s coefficient ( $\alpha$  in  $P(x) \propto x^{-\alpha}$ ), guiding the distribution’s decay rate. Our comparison revealed a higher exponent in column-count than in row-count, suggesting a faster decay and affirming the typical practice of constructing tables with rows for entities and columns for entity properties (dimension tables).

Using the `powerlab` library [Alstott et al., 2014], we observed exponents below 2 (e.g.,  $\alpha_{rc} \approx 1.5$  for row count), which is crucial since distributions with exponents under 2 lack well-defined mean or variance—a hallmark of true

<sup>8</sup><https://github.com/openai/tiktoken>

PassengerId	Survived	Pclass	Name	Sex	Age
1	0	3	Braund, Mr. Owen Harris	male	22.0
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
3	1	3	Heikkinen, Miss. Laina	female	26.0
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
5	0	3	Allen, Mr. William Henry	male	35.0
...	...	...	...	...	...
887	0	2	Montvila, Rev. Juozas	male	27.0

```

1 {
2   "github_repo": "mdmiqbal/Titanic-dataset",
3   "github_ref": "refs/heads/main",
4   "github_hash": "d80f02d",
5   "github_repo_path": "Assign 1(titenic data set 0).ipynb",
6   "extractor": "html",
7   "sourceline": 129,
8   "sourcepos": 8,
9   "before": "\n\n,\n      |\n\n    .dataframe tbody tr th:only-of-type {\n\n        vertical-align:\n↪ middle;\n\n          }\n\n    .dataframe tbody tr th {\n\n        \n↪ vertical-align: top;\n\n          }\n\n    .dataframe tthead th {\n\n        \n↪ text-align: right;\n\n            }";
10  "after": "\n\n891 rows x 12 columns";
11  "mime_type": "text/html",
12  "tar_path": "mdmiqbal-Titanic-dataset-d80f02d/Assign 1(titenic data set 0).ipynb"
13 }

```

[illegible]

Below is an example of a table which was classified as "Unknown" language. This particular table was entirely numeric, providing no language hints.

```
1 {
2   "github_repo": "nkaraffa/Intro-to-AI-Machine-Learning-and-Python-basics",
3   "github_ref": "refs/heads/main",
4   "github_hash": "c45317a",
5   "github_repo_path": "Classification_Model_Titanic.ipynb",
6   "extractor": "html",
7   "sourceline": 131,
```