The SpaseCroissant Oven: Automatic Metadata Generation For Open-Source Space Weather Datasets

by

Edenna H. Chen

B.S. Electrical Engineering and Computer Science, MIT, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

© 2025 Edenna H. Chen. This work is licensed under a CC BY-NC-ND 4.0 license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Edenna H. Chen
Department of Electrical Engineering and Computer Science
January 24, 2024

Certified by: Srini Devadas
Professor of Electrical Engineering and Computer Science, Thesis Supervisor

Certified by: Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair
Master of Engineering Thesis Committee

The SpaseCroissant Oven: Automatic Metadata Generation For Open-Source Space Weather Datasets

by

Edenna H. Chen

Submitted to the Department of Electrical Engineering and Computer Science on January 24, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

The rise of machine learning (ML) algorithms has led to a parallel rise in ML-ready datasets. A novel metadata schema released by OpenAI and MLCommons called Croissant, which is specifically designed for ML-ready datasets, aims to increase data accessibility, user understanding of data, and accuracy of claims based on data. However, current methods to automatically generate Croissant metadata present difficulties, such as involving manual entries. This can be especially difficult when attempting to preserve information about large ML-ready datasets, which are often derived from large scientific repositories belonging to organizations such as National Aeronautics and Space Administration (NASA). These major scientific repositories provide their own metadata standards, such as NASA's Space Physics Archive Search and Extract (SPASE) schema but context from this metadata can often be lost during data processing. This thesis presents a novel, improved approach to Croissant metadata generation which involves a hybrid parsing logic and Large Language Model (LLM) inference approach, as well as recommendations for future Croissant standards and SPASE to Croissant schema metadata conversion, that aims to retain this lost context.

Thesis supervisor: Srini Devadas
Title: Professor of Electrical Engineering and Computer Science

Thesis supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

The author would like to thank, first and foremost, God, because she can't do anything without Him. She would next like to thank her research supervisor, Dr. Jonathan P. How, for taking a chance on her and allowing her to work with space data for the first time. She would also like to thank her academic supervisor, Srini Devadas, and Jessica Zdon-Smith for their academic guidance throughout her years at MIT.

The author is also grateful for her parents, sister, cousin, and surrounding friends for being supportive and patient throughout this entire process. She would also like to thank the GeoCroissant working group and in particular GeoCroissant group lead Rajat Shande for his guidance, as well as MLCommons for letting her sit in on open meetings and observe.

Contents

List of Figures 9

List of Tables 11

1 Introduction: Contextualizing Data **15**

2 Related Work **19**

2.0.1 Croissant: A Metadata Schema for ML-Ready Datasets 19

2.0.2 Croissant Drawbacks 28

2.0.3 SPASE Schema 29

2.0.4 Existing Inference Methods and LLMs in Metadata Generation . . . 29

3 Use Case Curation **31**

3.0.1 Full STORM-AI Dataset 31

3.0.2 Test Cases 32

4 Methodology **35**

4.0.1 Overview 35

4.0.2 Frontend Design 37

4.0.3 Backend 40

5 Analysis **47**

5.1 Application Demonstration and Tutorial 47

5.2 SPASE XML to Croissant Metadata: Additional Methods 47

6 Future Work **49**

6.1 Suggestions for Improvement 49

6.1.1 User Interface & Frontend Design 49

6.1.2 Backend Logic 50

References 51

List of Figures

2.1 The Croissant lifecycle and ecosystem. [3] 20
2.2 File types supported by Croissant for dataset description and integration. . . 21
2.3 Croissant Metadata Properties and Descriptions 22
2.4 Croissant FileObject Metadata Properties and Descriptions. 23
2.5 Croissant FileSet Metadata Properties and Descriptions. 23
2.6 Croissant RecordSet Metadata Properties and Descriptions. 24
2.7 Croissant Field Metadata Properties and Descriptions. 25
2.8 Croissant Source Metadata Properties and Descriptions. 26
2.9 Croissant Data Types and Their Usage. 26
2.10 The MLCommons Croissant Editor UI. [8] 27
2.11 Examples of general functions available in the `mlcroissant` library. 28

4.1 Initial appearance of SPASECroissant frontend. 35
4.2 Directory structure of the SPASECroissant Flask application. 36
4.3 Color-marking the Description portion of Croissant metadata. 37
4.4 generate_croissant_async() function 39
4.5 mlcroissant library tutorial excerpt 39
4.6 SEction in which user interacts with with simplified workflow 40
4.7 Diagram of the Croissant metadata generation pipeline. 40
4.8 query_gpt() function 41
4.9 generate_dataset_map() function 41
4.10 Dataset map visualization. 42
4.11 SPASE and Croissant data types. 43
4.12 Prompt to generate the `mlc.Field description` field and classify data types. 44
4.13 generate_description() prompt 45

List of Tables

3.1 Test Cases for File Types 33

Acknowledgments

The author would like to thank, first and foremost, God, because she can't do anything without Him. She would next like to thank her research supervisor, Dr. Jonathan P. How, for taking a chance on her and allowing her to work with space data for the first time. She would also like to thank her academic supervisor, Srini Devadas, and Jessica Zdon-Smith for their academic guidance throughout her years at MIT.

The author is also grateful for her parents, sister, cousin, and surrounding friends for being supportive and patient throughout this entire process. She would also like to thank the GeoCroissant working group and in particular GeoCroissant group lead Rajat Shande for his guidance, as well as MLCommons for letting her sit in on open meetings and observe.

Chapter 1

Introduction: Contextualizing Data

Recent integration of artificial intelligence (AI) and machine learning (ML) into everyday technology has revealed that despite the abundance of existing digitized data, a major challenge in developing applicable machine learning models lies in formatting data for model training and design. Datasets that can be fed directly into model training pipelines, otherwise known as *ML-ready datasets*, are often created as a result of extensive manipulation, loss, or synthesis of information pulled from one or multiple original data sources.

The responsibility of transferring context and information on data development is often left to the user. Several initiatives within the machine learning community exist to promote this transfer of information, such as the FAIR dataset principles (Findable, Accessible, Interoperable, Reusable) in the scientific research and private sectors [1]. FAIR dataset principles encourage dataset curators to view datasets not as final products, but as continuation points of further research and model development by focusing on the following:

- **Findable.** Data should be easily indexed and searched for, and referenced using identifiers such as DOIs or UUIDs.
- **Accessible.** Data should be accessible using common standard protocols, with clear user instructions for access. If data is not accessible, metadata should be accessible.
- **Interoperable.** Data uses common, standardized terms and schemas. This allows data to be easily integrated into various models or systems.
- **Reusable.** Data should be given with long-term usability in mind. That is, context, description, and annotations– including information on sourcing, license, and provenance– should be given with the goal of maximizing longevity of considered data viability.

FAIR principles, developed in 2016, have already been incorporated into the initiatives of major organizations and data repositories. These parties include the Human Genome Project, the European Space Agency (ESA), the National Air and Space Agency (NASA), the World Bank, Google, and OpenAI [1].

Thus, data curators are encouraged to implement efforts to organize and provide as much context as possible with their collected data. These efforts often take the form of *metadata*: files that provide descriptions of their relevant datasets.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Spase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.spase-group.org/data/schema" xsi:schemaLocation="http://www.spase-group.org/data/schema http://www.spase-group.org/data/schema/spase-2_2_2.xsd">
  <Version>2.2.2</Version>
  <Instrument>
    <ResourceID>spase://SMWG/Instrument/CHAMP/Ephemeris</ResourceID>
    <ResourceHeader>
      <ResourceName>CHAMP Positions</ResourceName>
      <AlternateName>CHAMP Ephemeris</AlternateName>
      <ReleaseDate>2013-04-09T00:00:00Z</ReleaseDate>
      <Description>CHAMP spacecraft positions in various coordinate systems</Description>
      <Contact>
        <PersonID>spase://SMWG/Person/Robert.E.McGuire</PersonID>
        <Role>MetadataContact</Role>
      </Contact>
        </ResourceHeader>
    <InstrumentType>Platform</InstrumentType>
    <InvestigationName>Spacecraft position</InvestigationName>
    <ObservatoryID>spase://SMWG/Observatory/CHAMP</ObservatoryID>
  </Instrument>
</Spase>
```

Listing 1.1: Example of metadata. This metadata describes ephemeris data collected by the NASA satellite CHAMP. Its format and XML filetype aligns with the SPASE schema, a major defined metadata standard for space-related data.

However, using metadata to deliver context with datasets often comes with challenges that data curators are unable to overcome, including but not limited to [2]:

- There are a number of different metadata formats, *schemas*, particularly across different domains. Data curators are unsure of which schema to commit to when uploading a dataset.
- ML-ready datasets can contain data from multiple sources. In addition, each data source may use different assigned schemas. Data curators are unsure how to combine different kinds of schema into one metadata format, since ideally there would be one combined metadata file per dataset to reduce user workload.

- ML-ready datasets often contain *processed* data, a manipulated version of data from original sources. *This means that the original data source and the delivered ML-ready dataset are not the same.* This can result in user confusion regarding formatting, data extraction, and data meaning in the ML-ready dataset, since any data changes are not reflected in the metadata or documented properly elsewhere.

- Processed data may also result in loss of information regarding data errors or gaps. In particular, we find that data errors or gaps are sometimes not included in the original metadata for sources at all. Instead, this information may be located within multiple documents regarding the data source, which is often unbeknownst to the user or difficult to find.

In particular, we find that these metadata issues augment in major open-source repositories such as HuggingFace or Kaggle, where the prioritization of accessibility in machine learning and machine learning dataset curation becomes difficult to balance with FAIR datasets and ML models. Although the goal of incorporating more perspectives in the development of ML models is admirable, these repositories are extremely accessible to many users with a wide range of ML expertise. As a result, ML novices can present misleading claims regarding developed models trained with data that have missing context, such as how and where data were collected, the context in which the data was processed, and errors in the original data.

Thus, the question is: **How can we incorporate context, accessibility, and accuracy from original data sources when curating ML-ready datasets for public use?** This thesis attempts to answer this question by presenting a proof of concept pipeline for automatic metadata conversion and generation. This proof of concept utilizes use cases derived from an ML-ready dataset sourced from SPASE, a major space-related data repository and metadata schema. The presented use case generates new metadata that adheres to Croissant, a novel standardized schema for ML-ready data, while also including elements retained from the SPASE metadata schema. The pipeline is packaged in a locally-hosted application and incorporates novel amounts of inference in Croissant metadata generation through combining Large Language Model (LLM) querying and common parsing logic.

Improvements and suggestions for expanding the Croissant schema into more datatype domains are also presented in analysis of this project.

Chapter 2

Related Work

In the Introduction, we mentioned common challenges in contextualizing both original datasets and derived ML-ready datasets. We will now discuss previous work on these challenges, particularly work that this thesis will build on.

2.0.1 Croissant: A Metadata Schema for ML-Ready Datasets

First, we discuss a metadata standard that reflects FAIR principles called *Croissant*. Croissant is a novel metadata schema, first released in March 2024, that has since been integrated into major organizations and repositories. NeurIPS, a major ML, AI and computational neuroscience conference, encouraged participants in the Datasets and Benchmark track to use Croissant format "to document their datasets in machine readable way" at its December 2024 conference.

Croissant's development was led by MLCommons, in open collaboration with multiple parties including Google, King's College London, Harvard University, Meta, HuggingFace, NASA, and Kaggle. Croissant is designed to specifically describe ML-ready datasets, allowing datasets "to be loaded directly into ML frameworks and tools" such as TensorFlow, PyTorch, and JAX. Croissant is also incorporated into major dataset repositories such as Kaggle, HuggingFace, and OpenML. [3] As of March 2024, Google Dataset Search also offers a Croissant filter [4].

Croissant-compliant metadata "describes datasets' attributes, the resources they contain, and their structure and semantics", which "streamlines their usage and sharing within the ML community and between ML platforms and tools while fostering responsible ML practices".

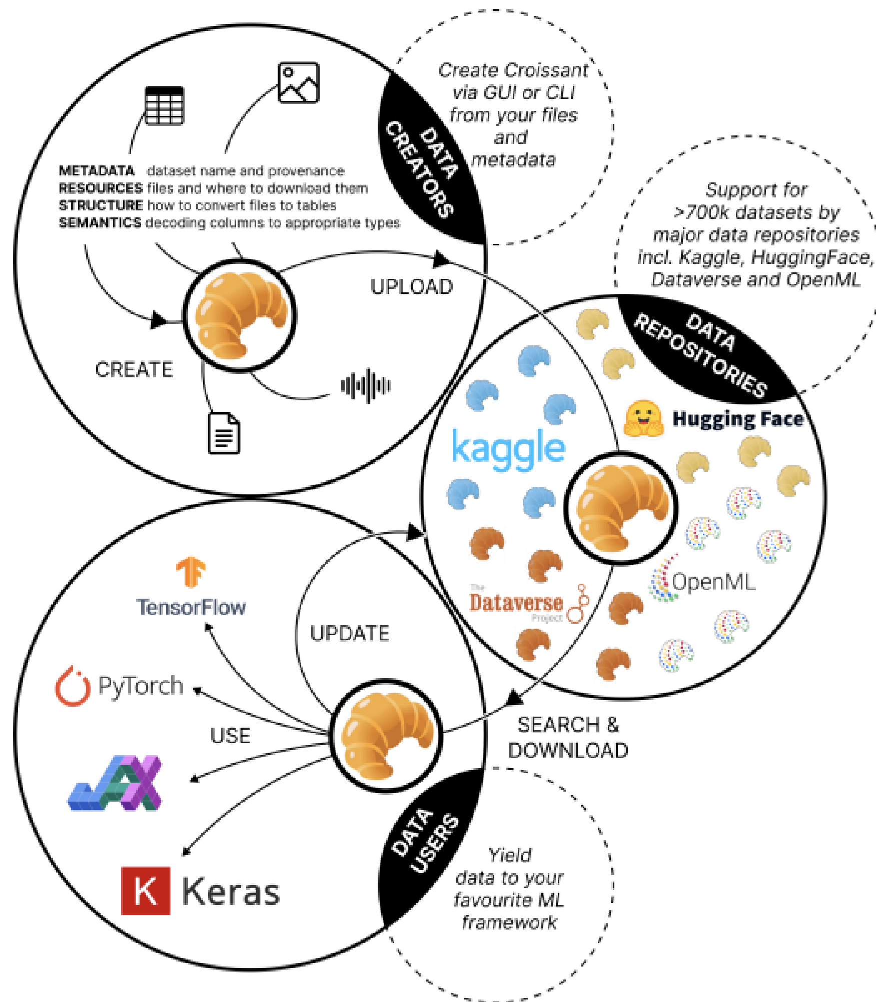

Figure 2.1: The Croissant lifecycle and ecosystem. [3]

Croissant Specifications

As of January 2024, Croissant metadata "can describe most types of data commonly used in ML workflows, such as images, text, audio, or tabular." In addition, while "datasets come in a variety of data formats and layouts, Croissant exposes a unified 'view' over these resources", allowing users to "add semantic descriptions and ML-specific information" without having to edit the dataset structure. This ensures easy integration of Croissant metadata into existing repositories and workflows.

File Type	Supported File Formats
Text Files	CSV, JSON, Plain Text
Image Files	JPEG, PNG, TIFF
Audio Files	MP3, WAV, FLAC
Video Files	MP4, AVI, MKV
Compressed Archives	ZIP, TAR

Figure 2.2: File types supported by Croissant for dataset description and integration.

Croissant metadata consists of four layers of information, hence the pastry-themed moniker, that describe the relevant dataset:

- *The Dataset Metadata Layer.* Crucial information (dataset name, description, version).
- *The Resources Layer.* Describes the original repository source.
- *The Structure Layer.* Describes and organizes the dataset structure.
- *The Semantic Layer.* Provides ML-specific data interpretation and semantics

These four layers exist as a nested hierarchical structure encoded in a JSON-LD file.

We will now further elaborate on the description of these layers and their attributes, which will provide a foundational understanding for our metadata conversion mapping.

The Dataset Metadata Layer. The Dataset Metadata Layer is the first layer in a Croissant metadata JSON file. Listing **??** displays what the Dataset-level information looks like.

```
1  {
2    "@context": {
3      "@language": "en",
4      "@vocab": "https://schema.org/"
5    },
6    "@type": "sc:Dataset",
7    "name": "simple-pass",
8    "conformsTo": "http://mlcommons.org/croissant/1.0",
9    "description": "PASS is a large-scale image dataset
      that does not include any humans ...",
10   "citeAs": "@Article{asano21pass, author = \"Yuki M.
      Asano and Christian Rupprecht and ...",
11   "license": "https://creativecommons.org/licenses/by
      /4.0/",
12   "url": "https://www.robots.ox.ac.uk/~vgg/data/pass/",
```

Listing 2.1: Example of the Dataset Metadata layer section.

We can break down the contents above Croissant layer into the following required Dataset-level information:

Property	**Expected Type**	**Cardinality**	**Comments**
`@context`	URL	ONE	A set of JSON-LD context definitions that make the rest of the Croissant description less verbose. See the recommended JSON-LD context in Appendix 1.
`@type`	Text	ONE	The type of a Croissant dataset must be `schema.org/Dataset`.
`dct:conformsTo`	URL	ONE	Croissant datasets must declare that they conform to the versioned schema: `http://mlcommons.org/croissant/1.0`.
`description`	Text	ONE	Description of the dataset.
`license`	CreativeWork, URL	MANY	The license of the dataset. Croissant recommends using the URL of a known license, e.g., one of the licenses listed at `https://spdx.org/licenses/`.
`name`	Text	ONE	The name of the dataset.
`url`	URL	ONE	The URL of the dataset. This generally corresponds to the Web page for the dataset.
`creator`	Organization, Person	MANY	The creator(s) of the dataset.
`datePublished`	Date, DateTime	ONE	The date the dataset was published.

Figure 2.3: Croissant Metadata Properties and Descriptions

The Resources Layer. The Resources Layer, encompassed in the `distribution` property, gives information on how the dataset is structured. This information expands past Schema.org standards in order to accommodate ML-ready datasets, which often have more nuanced layouts. The Resource Layer details dataset structure using two types of classes: the `FileObject`, which describes the individual files in the dataset, and the `FileSet`, which describes sets of files (e.g. in directories).

`FileObject` inherits from Schema.org's `CreativeWork`, and contains subproperties such as those included in Figure 2.4.

`FileSet` is an extension of Schema.org's `Intangible` property, and describes "collections of homogeneous files, such as images, videos or text files, where each file needs to be treated as an individual item, e.g., as a training example." [5] A `FileSet` exists within a container, which can be itself be a `FileObject`. A `FileSet` "may also specify inclusion / exclusion filters: these are file patterns that give the user flexibility to define which files should be

Property	Expected Type	Cardinality	Description
`sc:name`	Text	ONE	The name of the file. Reflects the downloaded file name including the extension, e.g., `"images.zip"`.
`sc:contentUrl`	URL	ONE	Actual bytes of the media object, e.g., an image or video file.
`sc:contentSize`	Text	ONE	File size in bytes (default) or specified units (KB, MB).
`sc:encodingFormat`	Text	ONE	The format of the file as a MIME type.
`sc:sameAs`	URL	MANY	URL (or local name) of a FileObject with the same content in a different format.
`sc:sha256`	Text	ONE	Checksum for the file contents.
`containedIn`	Text	MANY	Another FileObject or FileSet this file is part of (e.g., files extracted from archives). The `contentUrl` is evaluated as a relative path within the container.

Figure 2.4: Croissant FileObject Metadata Properties and Descriptions.

part of the FileSet. For example, include patterns may refer to all images under one or more directories, which exclude patterns may be used to exclude specific images." [5]

`FileSet` properties are described in Figure 2.5.

Property	Expected Type	Cardinality	Description
`containedIn`	Reference	MANY	The source of data for the FileSet, e.g., an archive. If multiple values are provided, the union of their contents is taken (e.g., combining files from multiple archives).
`includes`	Text	MANY	A glob pattern that specifies the files to include.
`excludes`	Text	MANY	A glob pattern that specifies the files to exclude.

Figure 2.5: Croissant FileSet Metadata Properties and Descriptions.

Note that `includes` and `excludes` properties use glob patterns to filter and identify the files that belong in FileSets; for example, "/foo/pic.jpg" or any ".jpg" image. The Croissant specification state that "to get the set of FileObjects included in the FileSet, the include pattern(s) are evaluated first. If multiple includes are specified, the union of their results is

taken. Then all the files corresponding to the excludes patterns are removed from that set. includes and excludes patterns are evaluated from the root of the containedIn contents (e.g., the top level directory extracted from an archive)." [5]

The Structure Layer. The Structure Layer gives information on the organization of the dataset content. This might sound similar to the Resources Layer, but we differentiate the two in that the Resources Layer describes the physical layout of the files in the dataset. Meanwhile, the Structure Layer describes the *logical layout* and *relationships* within the data. The Structure Layer accomplishes this with `RecordSet`, which "allows loading data of various formats into a standard representation, included structured (CSV and JSON) and unstructured (text, audio, and video) data." [3]

The Structure Layer accommodates the common state of "format heterogeneity" in ML-ready datasets. For example, a dataset might have a directory of images, with labels in a separate CSV file, and additional metadata in another separate JSON file. The Structure Layer `RecordSet` attribute describes how to link these pieces of data together in one mechanism. It does so using `RecordSet` subproperties such as `DataSource`, **??**, as well as `Extract` and `Transform`, both of which are contained within `DataSource` and describe where to access data within files or directories)

The Croissant specification defines a `RecordSet` as describing "a set of structured records obtained from one or more data sources (typically a file or set of files) and the structure of these records, expressed as a set of fields (e.g., the columns of a table). A RecordSet can represent flat or nested data." [5]

Within a `RecordSet`, the Croissant specification defines the `Field` type, which describes the data source and links it to what `FileSet` it uses: "`Field` is part of a `RecordSet`. It may represent a column of a table, or a nested data structure or even a nested RecordSet in the case of hierarchical data." [5] Note that Fields can reference other Fields in the dataset to join them together.

Property	Expected Type	Cardinality	Description
`field`	Field	MANY	A data element that appears in the records of the RecordSet (e.g., one column of a table).
`key`	Text	MANY	One or more fields whose values uniquely identify each record in the RecordSet.
`data`	JSON	MANY	One or more records that constitute the data of the RecordSet.
`examples`	JSON, URL	MANY	One or more records provided as example content of the RecordSet, or a reference to a data source that contains examples.

Figure 2.6: Croissant RecordSet Metadata Properties and Descriptions.

Property	**Expected Type**	**Cardinality**	**Description**
`source`	DataSource, URL	ONE	The data source of the field. Typically references a FileObject or FileSet's contents (e.g., a column of a table).
`dataType`	DataType	MANY	The data type of the field, identified by the URI of the corresponding class. This can be an atomic type (e.g., `sc:Integer`) or a semantic type (e.g., `sc:GeoLocation`).
`repeated`	Boolean	ONE	If `true`, the Field is a list of values of type `dataType`.
`equivalentProperty`	URL	MANY	A property equivalent to this Field. Maps specific fields to properties associated with the RecordSet's `dataType`.
`references`	Reference	MANY	Another Field from another RecordSet that this field references. Equivalent to a foreign key in a relational database.
`subField`	Field	MANY	Another Field nested inside this one.
`parentField`	Reference	MANY	A special case of `subField` that references an existing Field but remains hidden in the RecordSet.

Figure 2.7: Croissant Field Metadata Properties and Descriptions.

The Semantic Layer. Lastly, Croissant specifies the Semantic Layer, which helps describe ML-specific information about the dataset such as train, test, and validation dataset splits, labels, and common datatypes used in ML-ready datasets. Croissant specifies datatypes in ML-ready datasets include bounding boxes, categorial data, segmentation masks. Croissant can also take in dataTypes from other schemas, as long as the link to the specific schema data type is included to ensure usage of standardized, established schema definitions.

Property	**Expected Type**	**Cardinality**	**Description**
`fileObject`	Reference	ONE	The name of the referenced `FileObject` source of the data.
`fileSet`	Reference	ONE	The name of the referenced `FileSet` source of the data.
`recordSet`	Reference	ONE	The name of the referenced `RecordSet` source.
`extract`	Extract	ONE	The extraction method from the provided source.
`transform`	Transform	MANY	A transformation to apply on source data on top of the extraction, e.g., a regular expression or JSON query.
`format`	Format	ONE	A format to parse the values of the data from text, e.g., a date format or number format.

Figure 2.8: Croissant Source Metadata Properties and Descriptions.

Data Type	**Usage**
`sc:Boolean`	Describes a boolean.
`sc:Date`	Describes a date.
`sc:Float`	Describes a float.
`sc:Integer`	Describes an integer.
`sc:Text`	Describes a string.
`sc:ImageObject`	Describes a field containing the content of an image (pixels).
`cr:BoundingBox`	Describes the coordinates of a bounding box (4-number array). Refer to the section "ML-specific features > Bounding boxes".
`cr:Split`	Describes a RecordSet used to divide data into multiple sets according to intended usage with regards to models. Refer to the section "ML-specific features > Splits".

Figure 2.9: Croissant Data Types and Their Usage.

Croissant-Compliant Metadata Generation Tools

Automatic Generation on Kaggle & HuggingFace Currently, users may make an account on open-source repository sites Kaggle and HuggingFace. Once the accounts are made, Kaggle and HuggingFace allow users to generate Croissant files for any datasets in their

repositories. Neither of the websites make this generation code publicly available, although HuggingFace mentions dataset conversion to Parquet files in the process and that datasets must be public to generate Croissant files unless you have a professional account [6].

Croissant Editor MLCommons has released a visual editor for users to develop and modify Croissant-compliant dataset metadata by hand, as viewed in Figure **??**. Users require a HuggingFace account to access the editor, and infers information about dataset resources and RecordSets. The Croissant Editor aims to promote Responsible AI values in its use [3]. MLCommons has also made the code for the editor publicly available on GitHub [7]. The current Croissant Editor code does not incorporate LLM querying, which provides an unprecedent level of flexibility in metadata inferencing. Notably, the Croissant Editor also does not include inference support for generating FileSets from its recordSets, which could be an interesting area in which we could incorporate LLM querying.

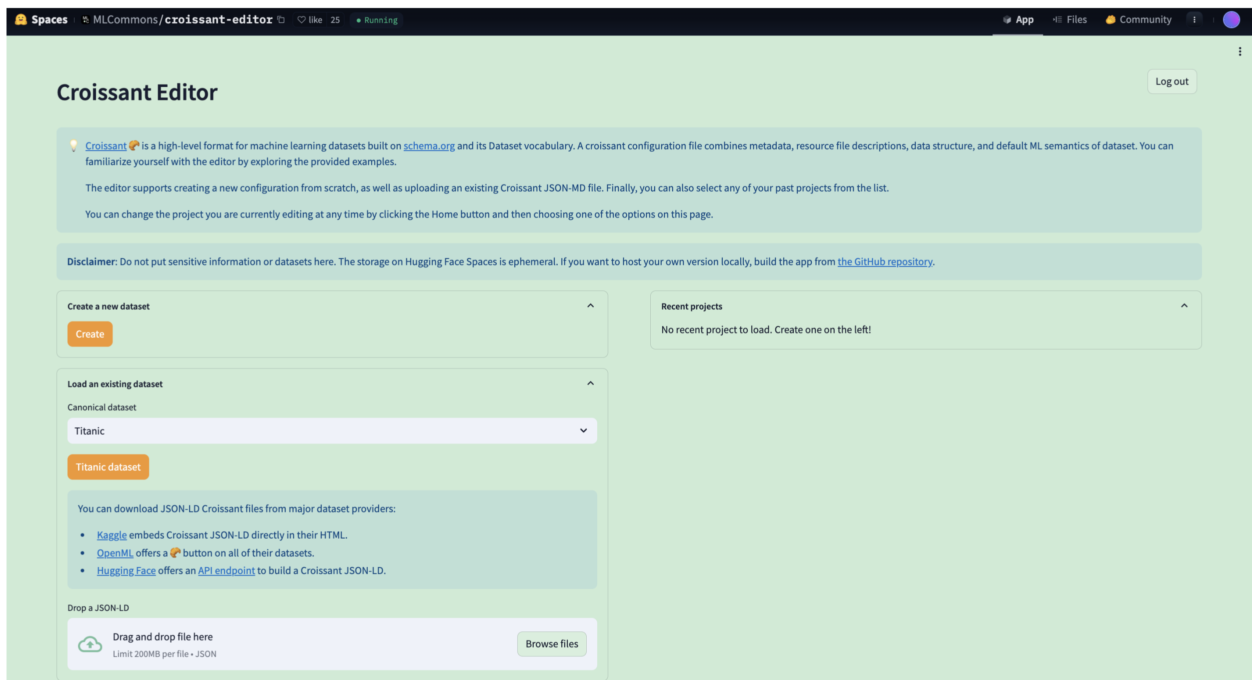

Figure 2.10: The MLCommons Croissant Editor UI. [8]

mlcroissant library Croissant offers a library, *mlcroissant*, which allows users to search, upload, and access data that uploaded to any repository with integrated Croissant metadata in their data storage. This includes the three aforementioned major open-source repositories: HuggingFace, Kaggle, and OpenML. In addition, *mlcroissant* offers the ability to build, parse, and validate Croissant metadata instances. Once a Croissant instance of a dataset is created, users may also easily access attributes and subattributes under the dataset. Figure 2.11 describes several major functions that the *mlcroissant* library offers.

Function	**Description**
`Dataset(jsonld: str)`	Initializes a `Dataset` object by loading Croissant metadata from a specified JSON-LD file or URL.
`metadata.to_json()`	Converts the dataset's metadata into a JSON-formatted string for easy inspection.
`records(record_set: str)`	Returns an iterator over the records in the specified record set, facilitating data loading and processing.
`verify()`	Validates the Croissant metadata to ensure it adheres to the schema and checks for completeness and correctness.
`save(filepath: str)`	Saves the Croissant metadata to a specified file path, allowing for easy sharing and storage.

Figure 2.11: Examples of general functions available in the `mlcroissant` library.

Croissant was developed to be flexible and modular regarding the variety in types of ML-ready datasets. Its specifications are meant to be expandable, particularly with supported filetypes and data types. However, *mlcroissant* has limitations on what filetypes and data types it can currently support. For example, mlc.DataTypes currently include the following: AUDIO_OBJECT, BOOL, BOUNDING_BOX, DATE, FLOAT, IMAGE_OBJECT, INTEGER, SPLIT, TEXT, and URL. MLCommons acknowledges the need for further work and library development to cover more dataset use cases.

2.0.2 Croissant Drawbacks

In analyzing the immense amount of work done regarding Croissant development, we find two areas of improvement. **The first area of improvement is that the current methods to automatically generate croissant metadata are to use the Croissant Editor or by automatically generating metadata via Kaggle and HuggingFace**. Via Croissant Editor, the frontend is not easily navigated by users and can be confusing to ML novices. Via Kaggle and HuggingFace, users must make an account on a site. And in both cases, the inferences done in generating the Croissant JSON-LD file could be more accurate, and smaller details and heavily nested attributes such as those contained in RecordSets or Fields could be made more accurate.

The second area of improvement is that when a user generates automatic Croissant in any method by just uploading a dataset, the user loses important metadata information from the original metadata and repository source. In other words, the original data source often has its own metadata that should also be included in handling any data processed from it. When a user drops only a dataset into the current Croissant metadata generators, this information about the original dataset is loss. It would be beneficial for users to include and transfer information from the original sources along with processed datasets.

2.0.3 SPASE Schema

Due to the nature of the dataset and test case curation, which will be discussed in the next chapter, we will use NASA's existing Space Physics Archive Search and Extract (SPASE Schema) as a pilot demonstration for metadata generation and conversion. SPASE is a major metadata standard used by NASA for space-related datasets. SPASE schemas come in XML, and more recently, JSON formats, and describe a variety of datasets. These datasets can contain the

Existing Schemas from major scientific communities, we will use NASA's SPASE files as a pilot demonstration for metadata generation and conversion. A fuller explanation of SPASE standards can be found at this HTML description for SPASE 2.7.0, the latest version release of the SPASE metadata format, but more specifically this thesis will be deriving information from ML-ready datasets that are associated with data sources that can be described with SPASE XML files. Specifically, this thesis will address to the `NumericalData`, `InstrumentType`, `Observatory`, and `People` fields often found in SPASE XML files.

2.0.4 Existing Inference Methods and LLMs in Metadata Generation

In 2024, researchers at Nagoya University converted 284 SPASE metadata records to the more generic Japan Consortium for Open Access Repository (JPCOAR) schema in order to increase visibility of SPASE datasets within the Japanese scientific community and on Google Dataset Search. To do so, they used Extensible Stylesheet Language Transformations (XSLT) to programmatically and specifically map specific elements from SPASE to JPCOAR. [9] Notably, LLMs and inferences were not involved here, which provided less flexibility between schema versioning and was more tedious in the method of programmatic mapping.

In addition, LLMs– specifically GPT-3.5 and GPT4o– have been used to annotate and supplement metadata generation in multiple contexts. Specifically, GPT-4 has been studied in domains such as extracting basic metadata information and schema mappings [10], where it was found that the model struggle with language specific to certain scientific domains. This research indicates that there should be a fine balance struck in metadata conversion, where LLMs may not be able to convert between *any* kind of metadata yet but can be specifically applied and *incorporated* in specific metadata conversions.

Recent GPT models have also been used to reduce metadata generation costs from manual cataloging costs [11], and have proven efficacy in handling structure data– specifically, tabular data [12]. GPT also proves effective in annotating data and converting various text files into JSON or XML format using methods like prompt engineering, hybrid querying and parsing approaches, and testing with different data from different specific scientific domains.

A common theme across these studies, however, mention GPT's difficulty in handing large inputs and domain-specific information. This indicates that there should be a fine balance struck in metadata conversion, where LLMs may not be able to convert between *any* kind of metadata yet but can be specifically applied and *incorporated* in specific metadata conversions. There is not significant research done on usage of LLMs in providing increased inference abilities in Croissant metadata conversion or generation.

Chapter 3

Use Case Curation

Here we will describe the process in which I developed and curated the use cases. This includes the dataset from which it as derived and the reasoning behind the testing process.

3.0.1 Full STORM-AI Dataset

The test cases for this thesis were pulled from the Satellite Tracking and Orbit Resilience Modeling with AI dataset (STORM-AI). The full dataset can be found at this public DropBox link, and was curated and developed for the second annual MIT ARCLab Prize for AI Innovation in Space, launched in November 2024. At the time of writing for this thesis, the competition has not yet concluded. [13]

MIT ARCLab Prize for AI Innovation in Space

For additional context, the challenge objective for the competition was as follows:

"The challenge objective is to develop cutting-edge AI algorithms for nowcasting and forecasting space weather-driven changes in atmospheric density across low earth orbit using historical space weather observations. The available phenomenology include solar and geomagnetic space weather indices, measurements of the interplanetary magnetic field, and measured solar wind parameters. Participants are provided with an existing empirical atmospheric density model and spacecraft accelerometer-derived in situ densities and are tasked with training or creating models to predict future atmospheric density values." [13]

In addition, Phase 1 of the competition gave participants the following instructions:

"Your objective is to design a model that, given a spacecraft's initial state and 60 days of space weather information directly preceding that state, can predict the next 3 days of atmospheric density values the spacecraft will observe. That is, your model should take in these inputs: a satellite's initial location, provided in both geodetic coordinates and orbital elements; space weather information for the 60 day period preceding the timestamp of the initial satellite location; X-Ray flux information for the 60 day period preceding the timestamp of the initial satellite location. Your model should then predict the sequence of orbit-averaged atmospheric density values that the spacecraft will observe in the future. This

prediction must span a period of 3 days directly following the timestamp of the initial satellite location." [13]

The full Phase 1 dataset includes ML-ready data processed from NASA and European Space Agency (ESA) repository sources, which both offer XML files to represent dataset metadata with. At the time of writing, it contained:

- **Initial state files:** contain samples of a satellite's initial orbital elements, geodetic (ITRF) positional coordinates, and a 5-digit file ID for each I/O pair in the provided training data. Each file name has the format [first file ID]_to_[last file ID]-initial_states.csv.

- **OMNI2 data folder:** Space weather information collected by NASA Space Flight Goddard Center and provided in 60-day segments (one 60-day OMNI2 history file per initial state). Each file name has the format omni2-[file ID]-[first day]_to_[last day].csv, where the dates are displayed as YYYYmmDD.

- **GOES data folder:** X-Ray flux information collected by NOAA'S GOES satellites and provided in 60-day segments (one 60-day GOES history file per initial state). Each file name has the format goes-[file ID]-[first day]_to_[last day].csv, where the dates are displayed as YYYYmmDD. Note: GOES data is not included in V3.0 of the Phase 1 dataset but will be released in V3.1

- **Thermospheric density data folder:** Time series orbit average density values collected by ESA satellites and provided in 3-day segments (one 3-day "forecasted" density file per initial state). Each file name has the format [spacecraft]-[file ID]-[first day]_to_[last day].csv, where the spacecraft is indicated by a 6-character designation and dates are displayed as YYYYmmDD.[13]

The author of this thesis was involved in the design, development, and curation of the full dataset over the course of five months, which included different derivations of ML-ready data for different stages of the competition. The test cases for this thesis were pulled from the OMNI2 portion of the public Phase 1 dataset, which contained the most columns (58) and diverse amount of data types.

3.0.2 Test Cases

Thus, I chose my test cases from the STORM-AI dataset, a real-life ML-ready dataset, to demonstrate the proof of concept pipeline.

The test cases are as follows:

The passing of these test cases encompass the baseline performance standards for my metadata conversion pipeline.

Originally I considered including a test case with only a SPASE XML or JSON file. This test case would have served as the base example conversion of a SPASE XML or JSON file. However, after further research, I concluded that since Croissant is meant to describe ML-ready datasets, it is more realistic to include test cases that include processed, ML-ready datasets. Thus, Test Case 3 includes the SPASE XML file as well as ZIP files.

Test Case	SPASE XML File	CSV File	.ZIP File
1			✓
2		✓	
3	✓		✓
4		✓	

Table 3.1: Test Cases for File Types

In addition, after further research, I concluded that given the difficulty in traversing the SPASE registry server and in minimizing user workload, it would also be interesting to include two test cases in which no XML files were uploaded. These two test cases explore increasing inference in the metadata generation process through LLMs. This improves upon the current inference capabilities of the current Croissant Editor, which are either nonexistent or nonflexible where they exist. Thus, in Test Case 4, there is no SPASE XML files included. In this test case, SPASE metadata information would still be included and transferred in the Croissant metadata, such as InstrumentType. Other sections such as Numerical Data are considered to be included in Croissant, with the RecordSets section of Croissant metadata covering dataTypes for both SPASE and Croissant. In addition, SPASE Observatory and InstrumentTypes are translated into the Distribution sections of Croissant, which will be described in the methodology section. The inference capabilities also include searching the existing SPASE registry for source guessing, relieving additional workload from the user.

The SPASE XML file I chose to use was the publicly available XML file for the NASA OMNI space weather dataset [14], downloaded from the public SPASE registry Github page. This XML file is associated with the source of the *omni2* section of the STORM-AI Phase 1 dataset, which includes processed data from the OMNI repository as detailed in the previous section.

The test CSV file is one of the split OMNI files from the STORM-AI Phase 1 dataset, namely `omni2-08118-20191101_to_20191231`.

The test ZIP file contains four of these OMNI2 files, which have the same data columns and format, from different time periods– a file set.

Each of the OMNI files contain 58 columns, with each column representing a different type of data involved in space weather. The columns include data types such as proton flux, F10.7 index, plasma spacecraft ID, and several kinds of spacecraft positioning coordinates.

Chapter 4

Methodology

Here we will describe the methodology and process by which the proof of concept was developed. We describe the metadata generation and conversion logic, the frontend design of the system, and the backend design of the system.

4.0.1 Overview

The final product consists of a frontend interface for user interactions which may take in a CSV file, a ZIP file, or a combination XML, CSV, and ZIP input. The system processes any of these three possible inputs using a combination of parsing logic and inference through LLM queries.

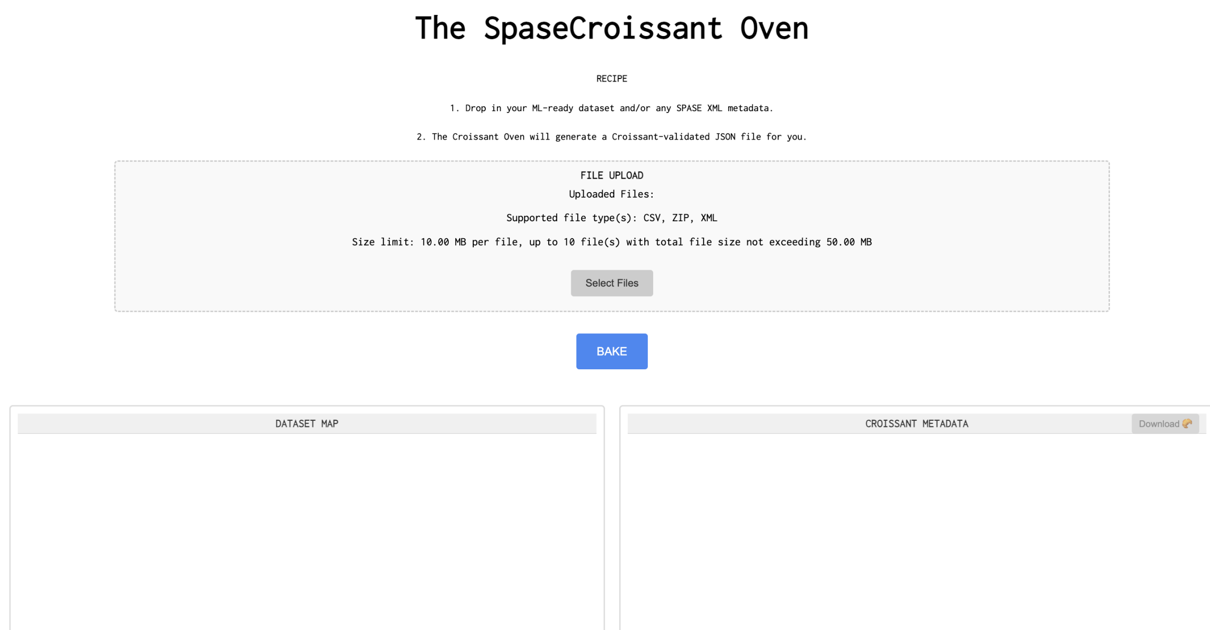

Figure 4.1: Initial appearance of SPASECroissant frontend.

After processing, two things are displayed. First, the generated Croissant metadata is

displayed on the bottom right. The generated Croissant metadata is rendered line by line and colored by section, which encourages additional user confirmation of the generated metadata.

Second, the input file mapping is displayed in the format of a text visualization of the compressed directory or CSV file. This provides an additional user sanity check that the correct data has been processed. In addition, the matching color between the mapping visualization and Croissant metadata provides additional benefits by increasing user understanding of what information is actually contained in the Croissant metadata and where the data structure is described, specifically under the *Distribution* section.

We will elaborate on and demonstrate the final product in the next section, but as a brief overview, we display the full system directory below:

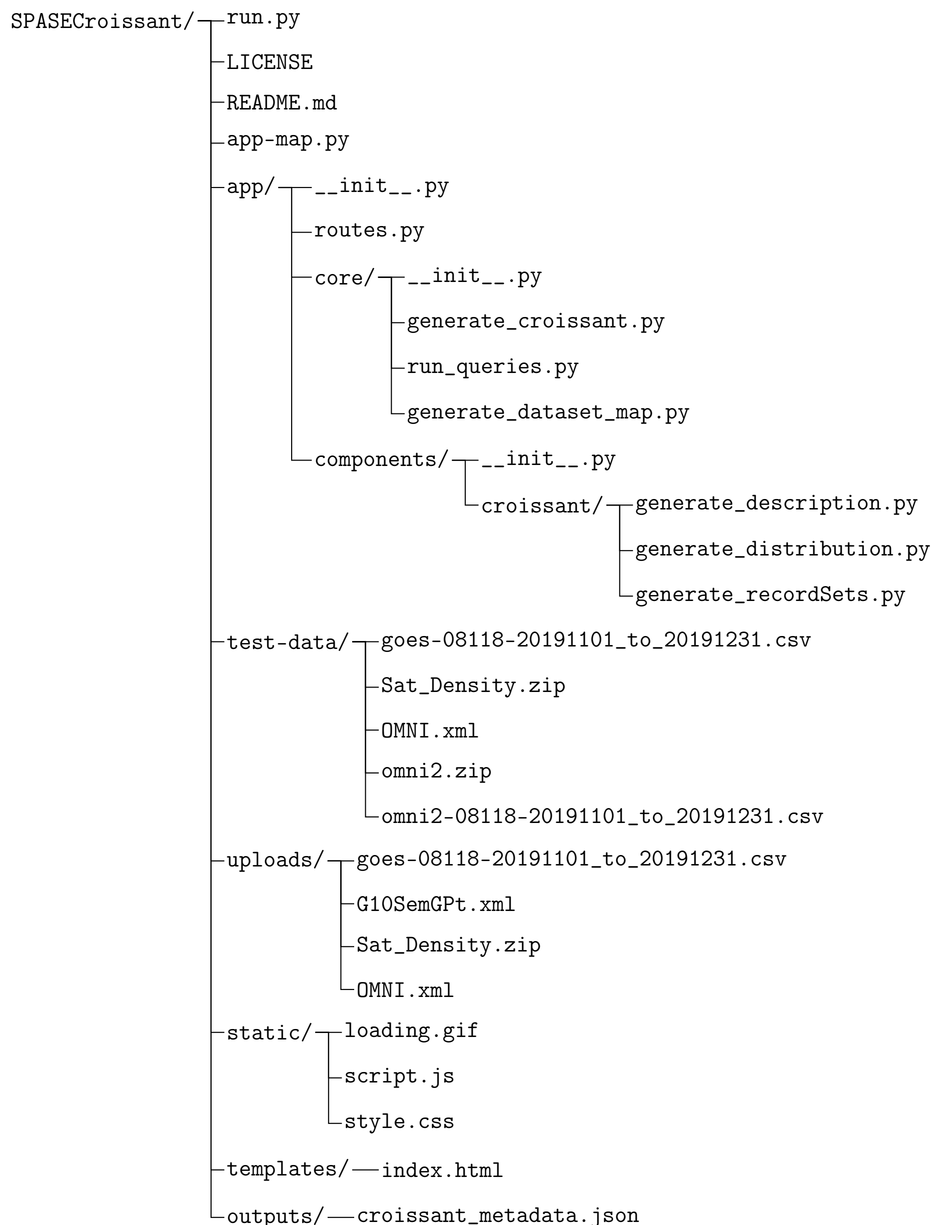

Figure 4.2: Directory structure of the SPASECroissant Flask application.

```
{
  "@id": "omni2-\\d{5}-\\d{8}_to_\\d{8}\\.csv-record_set/Scalar_B_nT",
  "@type": "cr:Field",
  "dataType": "sc:Float",
  "description": "spase:MeasurementType:MagneticField. The scalar value of the magnetic field in nanoteslas me
  "name": "Scalar_B_nT",
  "source": {
  {
    "extract": {
    {
      "column": "Scalar_B_nT"
```

Figure 4.3: Color-marking the Description portion of Croissant metadata.

4.0.2 Frontend Design

The interactive portion of the pipeline was arguably the most difficult section to build given my lack of experience in frontend design. I used Flask, a framework for web development in Python, to develop and test the product locally. The frontend design can be encompassed with the description of three files: `run.py`, `static/script.js`, `static/style.css`.

script.js

The JavaScript component of the frontend was arguably one of the most difficult for me. While outside resources were helpful in providing pointers or suggestions on how to implement subcomponents of the frontend, stitching it all together proved difficult.

One of the most tedious sections in the JavaScript component was the `.bake-button`, particularly since it was meant to trigger producing the and rendering output for the entire application. While it took some time, the backend was easier to work regarding having correctly formatted outputs. The most difficult part in the JavaScript component to work with was `renderTypingJson`, which visualizes the output of the Croissant metadata such that the text looks like it's being 'typed out', similar to how code or text renders on the ChatGPT interface.

The purpose of this rendering and Croissant metadata visualization styles serves two purposes.

First, as aforementioned, the user is more likely to trust, understand, and process the output of the Croissant metadata. From a human interaction perspective, if the text were to appear randomly, a user might find this disconcerting rather than feeling as if the inputted files actually took time to process and create. In addition, the loading GIF helps the user understand that the system *is* processing the file, and that it will take a bit of time– rather than being unaware of what is happening and being unsure if any files have been submitted or if the system has been triggered at all.

Either way, the backend process to process the files and generate the metadata remains the the same, but the feeling of a system taking the time to do so creates an increased sense of trust and understanding with the system. In addition, allowing the metadata to display gradually increases the changes of the user watching the metadata generate– and even if small, the odds of the user catching mistakes (because there is inference involved) and understanding the metadata format even marginally better

Secondly, the color-coding of the Croissant metadata visualization is meant to help enhance user understanding of the Croissant metadata generation process. The "description" sections of the metadata, which are one of the currently most heavily inferred pieces of information in the generated metadata, are shown in orange in order to highlight the inferred metadata and

allow easier confirmation and visual parsing by the user. Originally the generated Croissant JSON came out in a fully automatic-black block of text, which was much more difficult to understand, distinguish sections in, and manually parse, particularly since the Croissant metadata format includes nested information.

By rendering the metadata in this manner, I attempt to combat user negligence that can often comes from reliance on black-box workflows while also reducing user workload. The frontend application provides a simpler, more accurate workflow for conversion and metadata production, but it is still important for users to keep in mind that despite using better inference techniques like LLM querying and validation from the `mlcroissant` library, the system can still result in mistakes.

run.py

`run.py` contains the Flask logic for user interaction and utilizes crucial libraries such as `asyncio` for asynchronous operations and `mlcroissant` to build and validate the Croissant metadata generation process.

Upon user submission of appropriate files, `run.py` uses `process()` to concurrently process the files. `process()` intakes submitted files, downloads them to the `uploads` folder, and then concurrently runs the subprocesses to generate the text for the dataset map display and the Croissant metadata. Once both of these results are gathered, the information is returned in JSON format to the front end.

Another notable process in `run.py` is `generate_croissant_async()`, which awaits the return from three major components of Croissant metadata: distribution, recordSets, and the metadata description. Each of these components return lists of Croissant objects, or text, that are then pieced together `generate_croissant_async()` to create a full `mlc.Metadata` object.

Note that through the existing `mlcroissant` library, creating a `mlc.Metadata` object allows us to build a fully validated Croissant file. Here, I attempted to follow the mlcroissant tutorial as closely as possible. There were some slight changes I had to accommodate for due to Croissant specification updates and having to install the library from the Github source, such as additional dataTypes and subattributes for library objects like `Fields`.

The documentation for the library, including all updates, can also be tedious to parse across the different resources available describing `mlcroissant`, especially with additional updates. Thus, the tutorial Google Colab was extremely helpful in determining which attributes were available for each object.

While the current demonstration only shows the application processing CSV, XML, and ZIP files, I note that there is some included code that allows room for further expansion to other filetypes, specifically the ones that are allowed and available on the current Croissant Editor site. This room for improvement will be discussed under Future Work in Chapter 6.

The current Flask application pipeline runs locally. As seen in the `run.py` file, I ran the application on port 5001.

```python
async def generate_croissant_async(file_list):
    """Generate croissant metadata concurrently."""
    # try:
    # Gather distribution, recordSets, and description concurrently
    distribution = await generate_distribution(file_list)
    record_sets = await generate_recordSets(file_list, distribution)
    description = await generate_description(file_list, distribution, record_sets)

    print(f"Distribution: {distribution}", flush=True)
    print(f"Record Sets: {record_sets}", flush=True)

    metadata = mlc.Metadata(
        name="Dataset Metadata",
        description=description,
        cite_as=["Dataset Source"],
        url="http://example.com",
        distribution=distribution,
        record_sets=record_sets,
    )

    print(metadata.issues.report())
    print(metadata.to_json())

    return metadata.to_json()
```

Figure 4.4: generate_croissant_async() function

```python
def generate_dataset_map(file_list):

    def list_archive_contents(archive_path):
        """List contents of a zip archive in a hierarchical format."""
        contents = []
        with zipfile.ZipFile(archive_path, 'r') as zf:
            for file in zf.namelist():
                contents.append(f"        |_ {file.replace('/', '        |_ ')}")
        return "\n".join(contents)

    dataset_map = ["Dataset Map:"]
    for file in file_list:
        file = Path(file)  # Ensure file is a Path object
        # Check if the file is a zip archive
        if file.suffix == ".zip":
            dataset_map.append(f"- {file.name}")
            archive_contents = list_archive_contents(file)
            dataset_map.append(archive_contents)
        elif file.suffix != ".xml":
            # Include non-XML files directly
            dataset_map.append(f"- {file.name}")

    print("\n".join(dataset_map))
    return "\n".join(dataset_map)
```

Figure 4.5: mlcroissant library tutorial excerpt

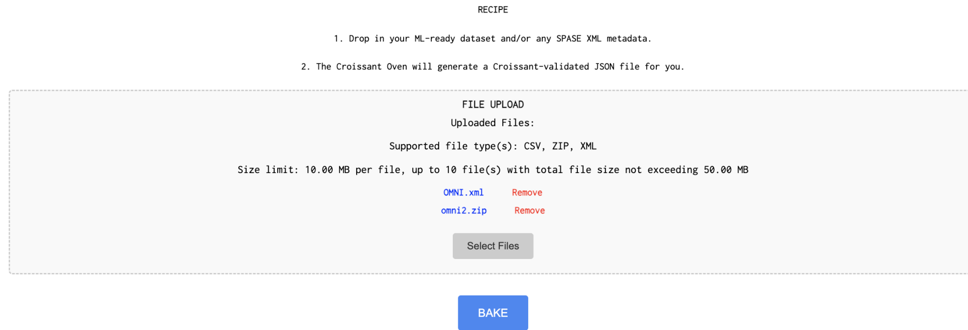

Figure 4.6: SEction in which user interacts with with simplified workflow

style.css

The CSS file for the page was relatively simpler to implement. I designed the frontend such that it would be as intuitive as possible for users to interact with, and tried to make the site similar to other free conversion sites available online. I attempted to keep the siteflow as simple as possible (all on the same page), and highlighted the BAKE button in order.

4.0.3 Backend

The backend consists of two parts: the generator for the dataset map representation, and the generator for the Croissant metadata JSON file. The pipeline for the Croissant JSON metadata, the main part of this project, is displayed below in ??.

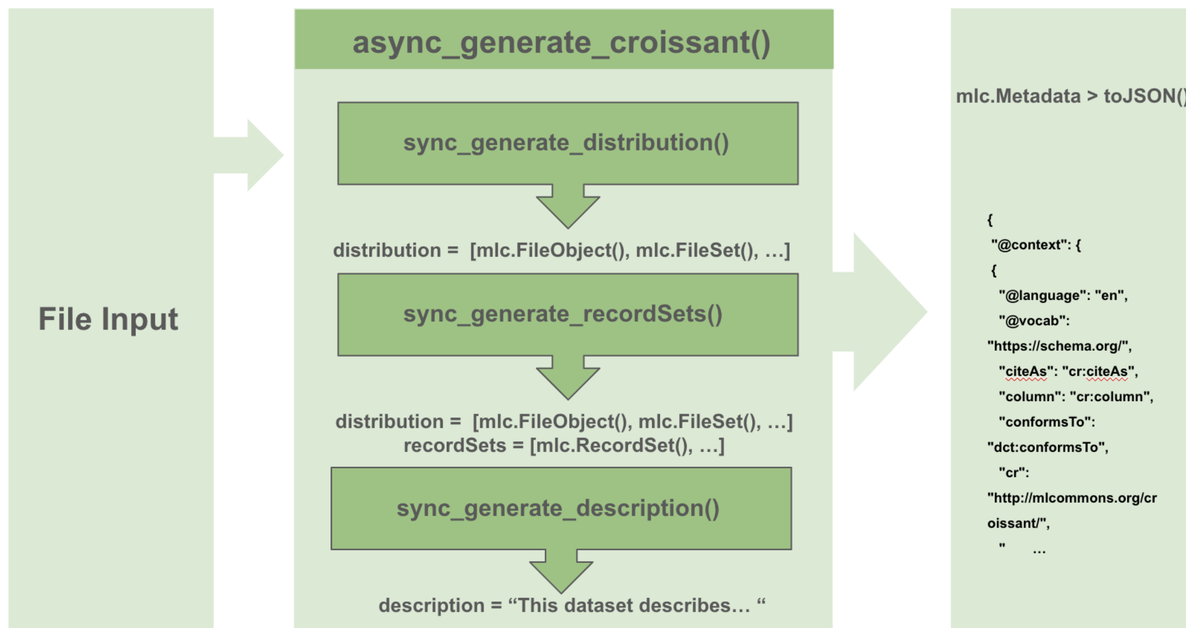

Figure 4.7: Diagram of the Croissant metadata generation pipeline.

```python
def query_gpt(prompt):
    """Query GPT-4 with a prompt and return the content of the response."""
    client = openai.OpenAI(api_key = "INSERTHERE")
    response = client.chat.completions.create(model="gpt-4", messages=[{"role": "user", "content": prompt}], max_tokens=4000)
    # Extracting and returning applicable textt
    message = response.choices[0].message.content
    print(message)
    return message
```

Figure 4.8: query_gpt() function

```python
def generate_dataset_map(file_list):

    def list_archive_contents(archive_path):
        """List contents of a zip archive in a hierarchical format."""
        contents = []
        with zipfile.ZipFile(archive_path, 'r') as zf:
            for file in zf.namelist():
                contents.append(f"        |_ {file.replace('/', '        |_ ')}")
        return "\n".join(contents)

    dataset_map = ["Dataset Map:"]
    for file in file_list:
        file = Path(file)  # Ensure file is a Path object
        # Check if the file is a zip archive
        if file.suffix == ".zip":
            dataset_map.append(f"- {file.name}")
            archive_contents = list_archive_contents(file)
            dataset_map.append(archive_contents)
        elif file.suffix != ".xml":
            # Include non-XML files directly
            dataset_map.append(f"- {file.name}")

    print("\n".join(dataset_map))
    return "\n".join(dataset_map)
```

Figure 4.9: generate_dataset_map() function

run_queries.py

One of the most notable parts of this project is the manner in which it incorporates inference into its metadata generation process. It does so by leveraging use of LLMs, specifically by using the OpenAI API to send queries to the ChatGPI 4o model. We will describe the querying process and how it plays into metadata inference in each subcomponent of `async_generate_croissant()`.

`run_queries.py` provides the function `query_gpt()`, which uses an OpenAI API key. You can find instructions on generating an OpenAI API key online [15]. It does require money and an account to pay for queries, but I note that the cost is fairly low and can be pay-by-use. The total cost for this project, with plenty of tokens to spare, was about $15 USD to run all tests.

One thing of note is that in order to accommodate for large queries, specifically ones in which the dataset had a large amount of columns and data types, the querying is used to

generate_dataset_map.py

`generate_dataset_map.py` provides the function `generate_dataset_map()`, which does not incorporate any LLM querying, but is a fairly simple parsing-based process. This text visualization is mostly aimed to visualize and allow user verification of contents in compressed files (zip, tar, gzip), so there is no need for inference here. Specifically for our test case, we leverage the metadata used in zip files that describes the file contents and structure without needing to open the zip file. This saves time and processing resources, allowing the `generate_dataset_map.py` process to run much faster than if the ZIP file was fully

```
Dataset Map:
- omni2.zip
        |_ omni2-08118-20191101_to_20191231.csv
        |_ omni2-08117-20191031_to_20191230.csv
        |_ omni2-08116-20191029_to_20191228.csv
        |_ omni2-08115-20191028_to_20191227.csv
```

Figure 4.10: Dataset map visualization.

unpacked.

generate_distribution.py

`generate_distribution.py` provides the function `sync_generate_distribution()`, which uses a combination of parsing logic and LLM querying to build a list of `mlc.FileObject()` and `mlc.FileSet()` items. At its simplest, files that are uniquely named, such as a singular CSV file or a ZIP file, are FileObjects. Meanwhile, if a compressed file like a ZIP file contains sets of files whose names can be encompassed with a regex pattern, this set of files becomes a FileSet.

`generate_distribution.py` incorporates the `mime` library, which is, in a sense, incorporating some inference into filetype guessing, but given the test cases we can still consider that the system currrently best handles CSV and ZIP files. The current `generate_distribution.py` also has several functions that have begun implementation for more filetypes, and the `mime` library should help handle future development in expanding `mlc.FileObject` and `mlc.FileSet` detection for additional filetypes.

generate_recordSets.py

The LLM and inference querying comes in at identifying dataTypes, since SPASE XML data tends to include extra datatypes. However, since the Croissant data has to be validated, we note that the SPASE datatypes are fitted into the `mlc.Field description` attribute, as well as a short sentence that the LLM delivers describing the data that that column describes. This is another major burden off of the user workload, as in the original editor and other automated Croissant generators on Kaggle and HuggingFace, the descriptions are not always automated or as easy to enter into the system. We can see how for a use case like the OMNI dataset, where there are 58 columns of data, it might be extremely helpful for some data description labelling to be inferred.

In addition to inferring the SPASE XML dataType, the inference portion of this function also identifies and infers the Croissant DataTypes to be placed into the `mlc.Field DataType` attribute. This is because in order for the `mlcroissant` library to validate the `mlc.Field` object, the `DataType` must be one of the allowed types in the current library. However, we are able to keep the SPASE data type intact in the metadata conversion process by moving

```
spaseMeasurementTypes = ["ActivityIndex",
                         "Dopplergram",
                         "Dust",
                         "ElectricField",
                         "EnergeticParticles",
                         "Ephemeris",
                         "ImageIntensity",
                         "InstrumentStatus",
                         "IonComposition",
                         "Irradiance",
                         "MagneticField",
                         "Magnetogram",
                         "NeutralAtomImages",
                         "NeutralGas",
                         "Profile",
                         "Radiance",
                         "Spectrum",
                         "ThermalPlasma",
                         "Waves",
                         "Waves.Active",
                         "Waves.Passive"
                        ]

croissantDataTypes = {"AUDIO_OBJECT": mlc.DataType.AUDIO_OBJECT,
                      "BOOL": mlc.DataType.BOOL,
                      "BOUNDING_BOX": mlc.DataType.BOUNDING_BOX,
                      "DATE": mlc.DataType.DATE,
                      "FLOAT": mlc.DataType.FLOAT,
                      "IMAGE_OBJECT": mlc.DataType.IMAGE_OBJECT,
                      "INTEGER": mlc.DataType.INTEGER,
                      "SPLIT": mlc.DataType.SPLIT,
                      "TEXT": mlc.DataType.TEXT,
                      "URL": mlc.DataType.URL
                     }
```

Figure 4.11: SPASE and Croissant data types.

it into the Croissant `mlc.Field description` attribute field, which is generated using a ChatGPT query.

One important note of developing `generate_recordSets.py` is that for the test case, the OMNI data contains 58 columns of different data types. This resulted in overflowing the maximum number of tokens allowed for GPT querying, which resulted in the solution of splitting any metadata passed into the prompt into two and conglomerating the two dictionaries back together– essentially, splitting one query into two and then combining the solution. The token for GPT querying is still set to 5000 to provide a higher upper cap for queries, but the metadata splitting is still performed in case datasets have a large amount of columns like the OMNI dataset test case.

generate_description.py

The last piece that we generate to build our full Croissant JSON file is `generate_description.py`, which uses the `distribution`, `recordSet` data, and XML file contents (if provided) to generate a 6-8 sentence description for the Croissant metadata.

One notable part of the prompt here that shows promise for additional research is the prompt to infer differences between the XML file and the processed ML-ready data. Another notable part of the prompt is that if there is no XML file submitted, the querying can be used to try and infer the SPASE dataset that the ML-ready data was pulled from if the user can't supply the original XML file or find it easily in the SPASE registry.

```python
def recordSet_prompt(metadata):

    prompt = f"""I have the following metadata for a dataset: {json.dumps(metadata, indent=2)}

                Here is a list of measurement types. We will refer to it as spaseMeasurementTypes:
                {spaseMeasurementTypes}

             Here is a list of Croissant data types. We will refer to it as croissantDataTypes:
             {croissantDataTypes.keys()}

             For each column, classify what measurement type it is out of types listed in spaseMeasurementTypes.
             For eah column, also classify what Croissant data type it is out of the types listed in croissantDataTypes.

             Then, you should generate and return ONLY a dictionary, with NO OTHER TEXT.
             DO NOT FORGET to output raw JSON only. DO NOT EVER include "json" at the beginning.
             DO NOT include any additional text, explanations, or formatting outside the JSON object.

             The dictionary should map each column name, in string format, to a list with two items.

             The first item in the list is a string in format "A. B." where:

             A is the string of URI representations for the column name's corresponding measurement type from Croissant data types.
             If it is found in the schema.org, the representation should be in the format of "sc:DateTime".
             If it is not found in schema.org, the representation should be in the format of "spase:MeasurementType:XXX",
             where XXX is explicitly one of the measurement types in spaseMeasurementTypes.

             B is a brief sentence describing what the column label means. IT MUST ONLY be ONE SENTENCE long.
             The sentence DOES NOT need to be grammatically correct. For example, you can say "Flags any excluded data points" versus
             starting with "This column flags" or "This column indicates".

             The second item in the list is the Croissant data type that the column is, in string format (exactly as listed in) croissant DataTypes.

            ### OUTPUT RULES ###
            - Your response MUST be valid JSON. DO NOT include any additional text, explanations, or formatting outside the JSON object.
            - DO NOT include "json" or any additional descriptors before the JSON object.
            - Ensure the JSON object is syntactically valid, properly enclosed with curly braces `{{}}` and all strings enclosed in double quotes `"`.
            - Return only raw JSON, nothing else.

            Here is an example of the expected format for your output:
            {{
            "Timestamp": [
                "sc:DateTime. The date and time when the corresponding data was collected or generated.",
                "DATE"
            ],
            "quad_diode": [
                "spase:MeasurementType:EnergeticParticles. The output measurement of a quad-bridged diode detector.",
                "FLOAT"
            ]
            }}
            """
    return prompt
```

Figure 4.12: Prompt to generate the `mlc.Field description` field and classify data types.

```
prompt = f"""
        Based on the provided dataset and record set information, as well as the parsed .xml data:

        1. Analyze and infer what changes were made between the dataset (from the distribution and recordSets) and the data detailed in the .xml file.
        2. Describe what specific information was taken from the data in the .xml file and how it is used.
        3. Identify any common mistakes or subtle details in the data from the XML file or the dataset that might be worth knowing or are not common knowledge.

        Here is the dataset distribution:
        {distribution_data}

        Here are the record sets:
        {record_set_data}

        Here is the parsed XML data:
        {xml_data}
        If the XML data is "NO XML DATA", please search the SPASE metadata registry and infer which dataset the dataset distribution and record sets are from.

        Return ONLY a 6-8 sentence description.
        2-3 sentences should be about the dataset source.
        The rest of the sentences should be about the structure of the data and things the user might want to note, given that it's an ML-ready dataset.
        Also include, at some point, the source URL of the dataset, either found in the XML or from your search in the SPASE metadata registry.

        The description MUST be written in an EXTREMELY decisive manner.
        You should only talk about your conclusions from the data.
        (ie, you should not say "based on" or use ANY non-deterministic language).

        For example, look at this description for reference on tone:

        The OMNI instrument is a virtual instrument representing the collection of Magnetometers, Electrostatic Analyers,
        and Energetic Particle instruments used in the creation of the OMNI datasets. The instruments used within the OMNI datasets
        come from the ACE, Wind, IMP8, Geotail, GOES and various other spacecraft. For any given time period OMNI uses a selection criteria
        to select the "best" available data to be used. Thus, only one magnetometer, one electrostatic analyser, and one energetic particle
        instrument are used for each data point within an OMNI dataset. Without looking at each data record it is not possible to tell
        which instruments contributed. As result, the OMNI instrument is used as a proxy for the set of possible contributing instruments.
        Complete details on the instruments and the selection criteria are available at the Information URLs listed below.

        """
```

Figure 4.13: generate_description() prompt

Chapter 5

Analysis

5.1 Application Demonstration and Tutorial

The instructions to download the Github code and run the program locally can be found at this linked Github repository. At its simplest form, one can clone the SPASECroissant repository. After inserting an OpenAI API key into the `run_queries` file, the user can then then enter the main directory and run the `run.py` file to get the site up and running locally. The test files are available in the GitHub repository as well, and an example video run for the ZIP and XML can be found at this Dropbox link.

The current OMNI files takes about 5 minutes to process each, which are much too long. In addition, the maximum upload capacity tested so far is about 2 MB. We will discuss maximum improvements in Chapter 6: Future Work, but the baseline standards for minimum viable product were reached as discussed in the introductory chapters.

5.2 SPASE XML to Croissant Metadata: Additional Methods

While the author was unable to change current Croissant metadata standards, we recall that this project aimed to provide a method and proof of concept by which SPASE XML and SPASE-based ML-ready data could be converted to Croissant-compliant metadata. Thee current proof of concept captures several aspects of SPASE data, including `dataTypes`, `NumericalData`, and additional instrument and observatory information from SPASE XML files (dependent on LLM querying). However, the author woud like to note additional observations made by which more thorough and exact SPASE metadata can be transferred into the Croissant metadata standards, as well as make some suggestions on the method by which this conversion can be more accurate.

Containing Observatory and Instrument Data in Croissant RecordSets

I suggest that `Observatories` could be contained in `RecordSets`, with the subattribute `Fields` containing the SPASE-specified `Instruments` and the relevant associated persons being detailed in the subsequent `Description` fields.

I also suggest that all Observatory and Instrument information be contained within the same RecordSet, so that there is only one RecordSet that contains the additional SPASE information to prevent extra confusion with multiple RecordSets. This RecordSet can be marked with a an id or hash, given that if additional SPASE XML or JSON files are provided, they may need to be saved in the ML-ready dataset in a certain way. This in itself could lead to debate over new formatting standards, but this is meant to be a Croissant-compliant, valid approach towards preserving as much information from the original data source metadata as possible.

Considering SPASE Numerical Data Attributes in Croissant FileSets

Within the `distribution` field, it may also be worth exploring placing SPASE `Numerical Data` attributes and information within the `FileSet` field of the Croissant format. SPASE data can often contain different types of files, which can result in different kinds of extraction or transformation methods (which are included field in the Croissant specification standards). Perhaps it would be helpful to also include libraries that are commonly used to unpack SPASE files such as NetCDF files.

Chapter 6

Future Work

6.1 Suggestions for Improvement

6.1.1 User Interface & Frontend Design

There are some minor changes that could be made on the application side. Firstly, my current application only runs locally, so it would be beneficial to make it more accessible and deploy it on a cloud service. In addition, I was unable to implement the Download button in which the generated Croissant file can be downloaded as a JSON. While the user can still copy-paste the output and it's formatted correctly as a JSON, it would be best to ensure the button works.

In addition, the current API has to be entered into the actual code itself for users to use the application, which means that users must have a ChatGPT account in order to make the involved queries. I can see two solutions to this problem. The first would be to find a querying service that doesn't require payment or accounts which would likely also mean lower quality LLM responses. The second would be to have the user

In addition, it might be nice to color code more of the Croissant file (for example, the full distribution section or the full recordSets section). This might be especially helpful if done in tandem with the dataset map visualization, for example: to show which sets of files correspond to which `fileSets` and which singular files correspond to which `fileObjects` in the `recordSets` section. This would provide an extra layer of understanding about the Croissant structure for the user.

Currently, the user is also unable to edit the generated Croissant file unless they have downloaded the file first. This is because Croissant validation is done in the backend process. It might be good to implement automatic Croissant validation or an editing box, where `mlcroissant` object and attribute suggestions might be included on the editing interface. This would be more complicated to implement, and the current Croissant Editor does provide manually input information. However, it might be beneficial for users to see which parts of the Croissant file they are actually editing to help them understand the content and structure of the metadata instead of black-boxing an output. It might also be beneficial to highlight the areas that were inferred or produced with LLM querying so the user is more inclined to confirm those areas of the files.

6.1.2 Backend Logic

Accommodating Additional Filetype & URL Links

Currently, given the test cases, the current application accommodates a mix of CSV, ZIP, and XML files. The current Croissant editor is able to support a higher variety of filetypes, as aforementioned in the Background in Chapter 2. The current Croissant Editor handles singlular filetypes– that is, files that aren't compressed archives– by converting them to `pandas` dataframes and processing them. The HuggingFace interface also mentions converting dataframes to parquet files prior to processing. It would likely be beneficial to explore both of these avenues in deciding how to process subsequent and additional filetype inputs.

Some SPASE XML files have also been recently updated to include a JSON version, so it would also be beneficial to expand input filetypes to SPASE JSON files as well instead of just XML files. These insights would be reflected in more diverse test cases, and specifically it would be best to test with the cases provided on the original Croissant Editor as well as additional SPASE data cases. We can consider that any ML-ready datasets derived from SPASE registry data sources would be appropriate to test with.

User Choice in LLM Inference Generation

I also thought it would be interesting to test with different LLM models that are available, although this would add an additional layer of complexity to the current GPT querying logic and ties back to the current issue of application efficiency.

It might also be interesting to experiment with more kinds of uploads specifically regarding metadata inference capabilities. We mentioned URL links, as ML-ready datasets can be found on Kaggle, HuggingFace, DropBox, Google Drives, and more cloud-based storage services. However, many HuggingFace datasets include official public papers in their descriptions, some of which provide extra credibility to the dataset due to

Increasing Efficiency

The current application has resource limitations on the maximum column types; as aforementioned, the uploaded OMNI test cases process quite slowly due to the 58 columns of datatypes that they include. Thus, it would be beneficial to add more concurrent processing methods so that the metadata is generating faster.

References

[1] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. *The FAIR Guiding Principles for scientific data management and stewardship*. Nature Publishing Group: Scientific Data, Mar. 2016. URL: https://www.nature.com/articles/sdata201618 (visited on 12/29/2024).

[2] Unknown. *Repository Metadata: Approaches and Challenges*. Johns Hopkins University: JScholarship, Aug. 2010. URL: https://jscholarship.library.jhu.edu/bitstream/handle/1774.2/33517/RepositoryMetadata_CCQ_FINAL.pdf (visited on 12/29/2024).

[3] M. Akhtar et al. *Croissant: A Metadata Format for ML-Ready Datasets*. Cornell University: arXiv, Mar. 2024. URL: https://arxiv.org/abs/2403.19546 (visited on 12/29/2024).

[4] G. Research and MLCommons. *Croissant: A Metadata Format for ML-Ready Datasets – Supported Frameworks and Tools*. Google LLC: Google Research Blog, Mar. 2024. URL: https://research.google/blog/croissant-a-metadata-format-for-ml-ready-datasets/ (visited on 12/29/2024).

[5] MLCommons. *Croissant Metadata Specification*. https://docs.mlcommons.org/croissant/docs/croissant-spec.html#dataset-level-information. Accessed: 2024-12-31. 2024.

[6] H. Face. *Croissant Dataset Viewer Documentation - Hugging Face*. https://huggingface.co/docs/dataset-viewer/en/croissant. Accessed: 2024-12-31. 2024.

[7] MLCommons. *Croissant Editor - MLCommons GitHub Repository*. https://github.com/mlcommons/croissant/tree/main/editor. Accessed: 2024-12-31. 2024.

[8] MLCommons. *Croissant Editor - MLCommons on Hugging Face*. https://huggingface.co/spaces/MLCommons/croissant-editor. Accessed: 2024-12-31. 2024.

[9] CODATA Data Science Journal. "Data Science Journal, Article 040 (2024)". In: *Data Science Journal* (2024). Accessed: 2025-01-24. DOI: 10.5334/dsj-2024-040. URL: https://datascience.codata.org/articles/10.5334/dsj-2024-040?utm_source=chatgpt.com.

[10] Anonymous. *Enhancing Metadata Extraction Using Large Language Models (LLMs)*. arXiv preprint. https://arxiv.org/abs/2310.11318. 2023.

[11] Anonymous. *Web Archives Metadata Generation with GPT-4o*. arXiv. https://arxiv.org/html/2411.05409v1. 2024.

[12] A. Unknown. "Exploring Large Language Models for Structured Data Parsing". In: *Proceedings of the 2024 Structured Data Processing Conference*. Accessed: 2025-01-24. 2024. URL: https://aclanthology.org/2024.sdp-1.14.pdf.

[13] *2025 AI Challenge Documentation*. Accessed: January 24, 2025. URL: https://2025-ai-challenge.readthedocs.io/en/latest/README.html.

[14] H. S. M. W. Group. *OMNI Instrument XML*. https://github.com/hpde/SMWG/blob/master/Instrument/OMNI.xml. Accessed: 2025-01-21.

[15] M. Development. *How to Use a ChatGPT API Key*. Accessed: 2025-01-24. 2025. URL: https://www.merge.dev/blog/chatgpt-api-key.