Dataverse: Open-Source ETL (Extract, Transform, Load) Pipeline for Large Language Models

Hyunbyung Park[1], **Sukyung Lee**[2], **Gyoungjin Gim**[2]
Yungi Kim[3], **Dahyun Kim**[4], **Chanjun Park**[5†]

[1]Moreh, [2]Upstage AI, [3]Liner, [4]Twelve Labs, [5]Korea University
hyunbyung.park@moreh.io, {sukyung, gyoungjin.gim}@upstage.ai
eddie@linercorp.com, kian.kim@twelvelabs.io
bcj1210@korea.ac.kr

Abstract

To address the challenges associated with data processing at scale, we propose Dataverse[1], a unified open-source Extract-Transform-Load (ETL) pipeline for large language models (LLMs) with *a user-friendly design* at its core. Easy addition of custom processors with block-based interface in Dataverse allows users to readily and efficiently use Dataverse to build their own ETL pipeline. We hope that Dataverse will serve as a vital tool for LLM development and open source the entire library to welcome community contribution. Additionally, we provide a concise, two-minute video demonstration of our system, illustrating its capabilities and implementation[2].

1 Introduction

The success of large language models (LLMs) is widely attributed to the scale of the data (Zhao et al., 2023), otherwise known as the *'scaling law'* (Kaplan et al., 2020) where LLM performance directly correlates with data size. Consequently, there has been an exponential growth in the need for massive data to further fuel LLM development. Such increase in demand leads to more complex data processing pipelines, as even simple operations need to be optimized for data processing at enormous scales. To handle such data workloads efficiently and effectively, distributed systems and techniques such as Spark (Zaharia et al., 2016) and Slurm (Yoo et al., 2003) have become crucial.

Unfortunately, the existing open-source data processing tools based on distributed systems (Mou et al., 2023; Soldaini et al., 2024; Lee et al., 2022a; Penedo et al., 2024) either lack easy customization support or a wide variety of operations such as deduplication (Xia et al., 2016), decontamination (Yang et al., 2023), bias mitigation (Shrestha et al., 2022), and toxicity reduction (Wang and Chang, 2022). This forces researchers to undergo a steep learning curve or cobble together tools from various sources, hindering efficiency and user experience.

In response to these limitations, we present Dataverse, a *unified* open-source ETL (Extract, Transform, Load) pipeline with *a user-friendly design* that enables easy customization. Inspired by the Transformers library (Wolf et al., 2019), Dataverse is built with a design principle of minimizing complex inheritance structures. Such design choice allows for easy addition of custom data operations. Specifically, the ETL pipeline in Dataverse is defined by block-based interface, which enables intuitive customization of ETL pipelines by simply adding, removing, or reshuffling blocks. Further, Dataverse natively supports a wide range operations needed to cover diverse data processing use-cases.

Moreover, the data processing workloads can be distributed among multiple nodes with Spark by simply setting the necessary configurations. Further, user-friendly debugging features via Jupyter notebooks are included for fast build-test of custom ETL pipelines. In addition, Dataverse supports multi-source data ingestion from on-premise storage, cloud platforms, and even web scraping. This feature empowers users to easily transform raw data from various sources. Driven by the aforementioned features, we posit that Dataverse will be a useful tool for effortlessly building custom ETL pipelines at scale for fast LLM development.

2 Why Dataverse?

In the era of LLMs, data scales exponentially (Kaplan et al., 2020), necessitating an efficient and scalable solution (Wang et al., 2023). Not only

† Corresponding Author
[1]https://github.com/UpstageAI/dataverse
[2]https://www.youtube.com/watch?v=yYyyLuPNK5s&t=33s

Open-Source Library	Dist. System	Expandable	Customization Difficulty
text-dedup	Spark	✗	N/A
DPS	Spark	✗	N/A
deduplication-text-datasets	Rust	✗	N/A
Dolma	Rust	✗	N/A
Datatrove	Slurm	O	High
Dataverse	Spark	O	Low

Table 1: Comparison between existing open-source LLM data processing libraries and Dataverse. "Dist. System", "Expandable", "Customization Difficulty" indicate the distributed system integrated into the library, whether the library is designed to be future-proof and capable of growth, and the difficulty of the customization, respectively. N/A means customization is not natively supported.

that, the fast pace of the LLM literature comes with the need to support a wide range of data operations such as toxicity and bias removal (Garg et al., 2023), personally identifiable information (PII) obfuscation (Schwartz and Solove, 2011), and data quality filterings (Shin et al., 2022; Choi and Park, 2023). Thus, on top of utilizing distributed systems, LLM-aware data processing also requires natively supporting a wide variety of operations and easy addition of custom data operations.

While there are many existing data processing libraries such as proposed (Mou et al., 2023; Soldaini et al., 2024; Lee et al., 2022a; Penedo et al., 2024), none of them are not yet the whole package of being easy to customize and supporting a wide variety data operations. To address this gap, we introduce Dataverse with a user-friendly design in mind, allowing users to utilize (custom) data processing tools and distributed system via simply setting the blocks and configurations. In the following sections, we detail the comparison between Dataverse and other existing open-source frameworks for LLM-aware data processing.

2.1 Comparison Between Dataverse and Other Open Source Libraries

As explained in the previous section, data processing libraries for the LLMs need to support a wide variety of data operations and distributed systems for scalable data processing. Further, the library itself needs to be expandable to accommodate novel data processing operations as they emerge. Lastly, taking one step further from just being expandable, it would be ideal if such expansion to custom data processing operations can be made effortlessly. We compare various open source data processing libraries and Dataverse in the aforementioned criteria in Table 1.

As shown in the table, the compared open source libraries, including text-dedup (Mou et al., 2023), DPS[3], deduplicate-text-datasets (Lee et al., 2022a), Dolma (Soldaini et al., 2024), and Datatrove (Penedo et al., 2024), as well as Dataverse all support distributed system. Specifically, text-dedup, DPS, and Dataverse use Spark (Zaharia et al., 2016) as the distributed system of choice, while deduplication-text-datasets utilizes parallel processing of Rust (Matsakis and Klock II, 2014) and Datatrove uses Slurm (Yoo et al., 2003) for their distributed system.

The comparison becomes clearer once we look at the "expandable" criteria. **Expandable** means rather than being a static library provided **as is**, rather dynamic, evolving library inherently designed to grow and adapt over time. Specifically, libraries such as text-dedup, deduplication-text-datasets, and Dolma all lack expandability as they are developed for one-time use, for instance, academic purposes. In contrast, Datatrove and Dataverse both support expanding the library suitable for future-proof LLM data handling. They feature interfaces that facilitate ongoing modification of the library. Also, they encourage community engagement by providing guidelines and processes for contribution ensuring the library remains adaptable and up-to-date.

Further, we compare how difficult it is to customize the library for a user's data processing workload. Note that for libraries that are not expandable, comparing the customization difficulty is not applicable. The customization difficulty for Datatrove is high as a user needs to make code changes to multiple places while adhering to the complex inheritance design of the Datatrove library. Conversely, the customization difficulty for Dataverse is low as the user simply needs to define a custom data processing operation function and register it to the

[3] `https://github.com/EleutherAI/dps`

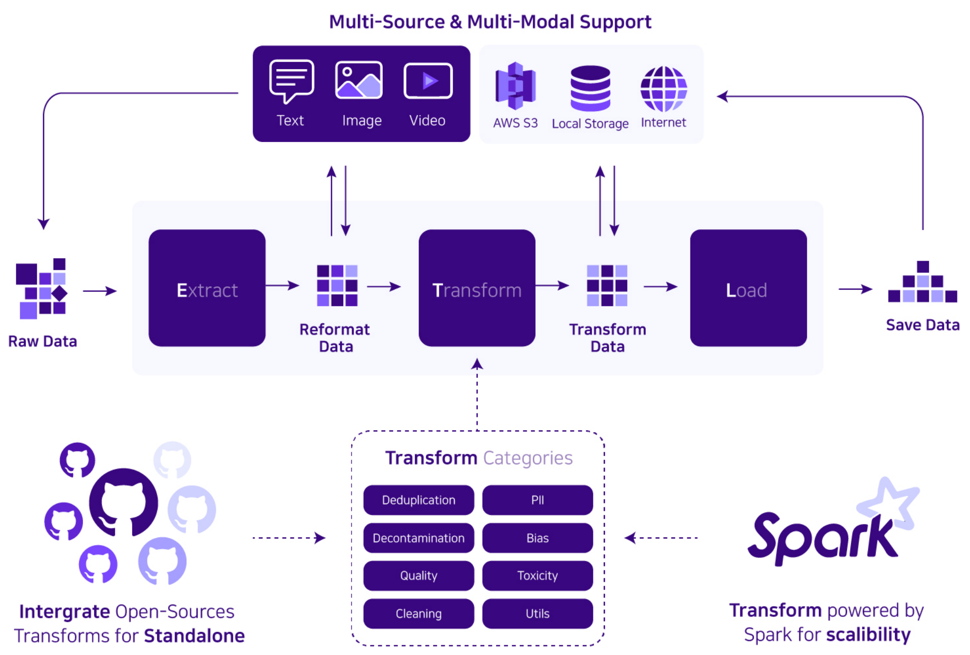

Figure 1: Overview of the Dataverse library.

Dataverse library using a decorator, as illustrated in Section 3.3. We now explain the key features and system architecture of Dataverse in the following sections.

3 Dataverse

Dataverse is an open-source library for building ETL pipeline for LLMs with user-friendly design at its core. The overview of the Dataverse library is shown in Figure 1.

3.1 Key Features

User-friendly design. The user-friendly design of Dataverse is implemented in consideration to various aspects. First, various tools necessary for building a complete ETL pipeline are optimized and unified such that users can use Dataverse as a standalone solution for building their custom ETL pipelines. As such, Dataverse natively supports optimized functions for various steps in the data processing workflow such as data downloading, reformatting, processing, and storing, ridding the need to look for other solutions even at very large data scales. Detailed explanation on the supported functionalities is given in Section 3.2.

Second, to support easy customization of ETL pipelines, Dataverse incorporates a strikingly simple method of adding custom data processing functions via Python decorators. Thus, users can readily utilize custom functions beyond that of the already large number of natively supported operations that are registered.

Third, utilizing either the natively supported operation or an custom added function to create an ETL pipeline in Dataverse is intuitive and flexible. The reason is that ETL pipelines in Dataverse are implemented using a block-based interface such that users can define a modular *block*, an atomic unit of data processing. Then, users can change their ETL pipeline by re-organizing the defined blocks, allowing for straightforward development of data processing pipelines. Further, Dataverse supports local testing functionality via Jupyter notebooks which allows users to inspect their ETL pipeline at various stages before scaling out.

Scalability via Spark and AWS Integration To scale ETL pipelines efficiently, Dataverse leverages Apache Spark (Zaharia et al., 2016), enabling distributed processing capabilities. Furthermore, it natively integrates with Amazon Web Services (AWS) for cloud utilization, facilitating greater scalability. Currently, Dataverse supports AWS S3 for cloud storage and Elastic MapReduce (EMR) for data processing. This integration ensures that users without access to sufficient local computing resources can effectively manage their data without encountering steep limitations. The aforementioned features can be enabled by simply changing the configuration or giving an argument when running the ETL pipeline.

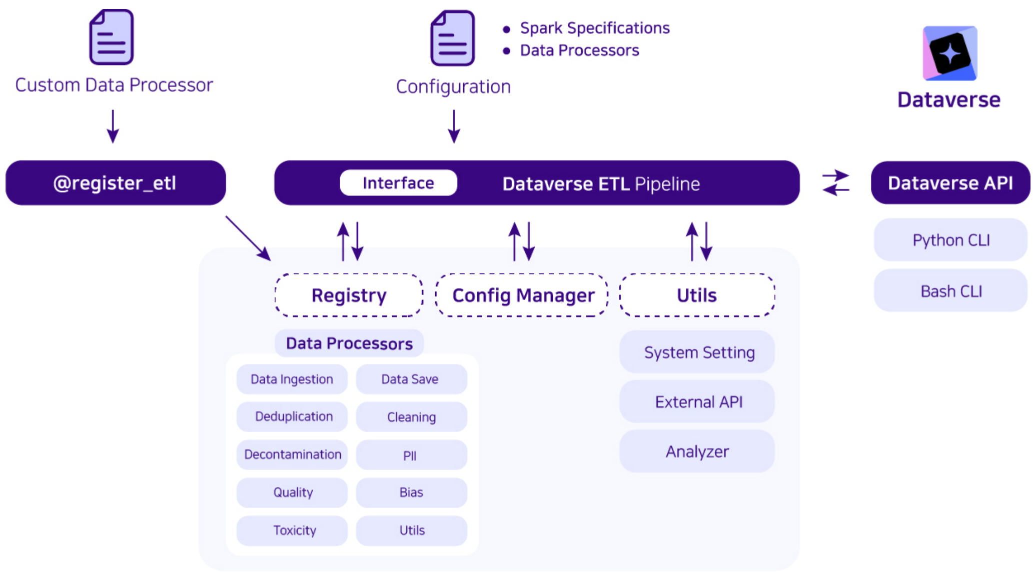

Figure 2: Architecture of Dataverse.

3.2 System Architecture

Figure 2 illustrates the overall system architecture of Dataverse.

ETL pipeline. The ETL pipeline represents the primary interface for Dataverse users. This central core interface facilitates communication with various modules, including configuration, registry, application programming interface (API), and utilities. Its primary objective is to ensure the seamless creation and operation of the ETL pipeline, effectively managing data processing tasks. Additionally, the interface offers AWS EMR integration by simply passing the "True" value to the "emr" option, as described in Section 3.3. This straightforward approach empowers users to leverage the scalability of cloud computing without the steep learning curve typically associated with distributed systems management.

Configuration. The user prepares a configuration object that encapsulates all the essential details required to execute the ETL Pipeline. The configuration facilitates the setup of Apache Spark specifications and the selection of the data processors to be employed.

Configuration manager. The configuration manager manages various configurations from specified paths (local, AWS S3) or handles multiple types (Python Dict, YAML, OmegaConf) of configuration data. It converts these configurations into a unified format compatible with Dataverse, ensuring they are ready for use in the system.

Registry. The registry serves as a repository where all data processor functions are stored. The data processors to be utilized are specified within the configuration which are then retrieved from the registry to assemble the desired ETL pipeline. Note that custom data processors can be added by simply using the `@register_etl` decorator. The list of natively supported data processors is as follows:

- **Data Ingestion**: Facilitating the loading of data from various sources (*e.g.*, data in Huggingface Hub, and parquet/csv/arrow format data in local storage) into a preferred format.
- **Data Saving**: Persisting the processed data into a preferred destination, such as a data lake or database.
- **Deduplication**: Eliminating duplicated data on dataset by dataset basis or globally across multiple datasets.
- **Data Cleaning**: Removing irrelevant, redundant, or noisy information from the data, such as stop words or special characters.
- **Data Decontamination**: Identifying and removing contaminated data such as benchmark datasets.
- **Personally Identifiable Information (PII) Removal**: Ensuring the removal of sensitive information, such as personally identifiable data, from the dataset.

- **Data Quality Enhancement**: Improving the quality of data from the perspectives of accuracy, consistency, and reliability for LLMs.
- **Bias Mitigation**: Reducing skewed or prejudiced data, with a particular emphasis on data that reinforces stereotypes of LLMs.
- **Toxicity Removal**: Identifying and eliminating harmful, offensive, or inappropriate content within the data.
- **Utilities**: Providing essential functionalities for data processing, including sampling, logging, and statistical analysis.

Utilities. The Utilities module serves as an internal helper toolset. One of its core features is the API utilities, which streamline the use of various external APIs such as AWS EMR. It simplifies the deployment and management of AWS EMR, reducing the complexity for researchers unfamiliar with cloud infrastructure. By simply setting their own AWS Credentials, Dataverse automatically handles the intricate details of provisioning EMR clusters and orchestrating the data processing pipelines across the cluster nodes.

Dataverse API. The Dataverse API serves as a gateway for users. Currently, Dataverse supports the Python CLI (Command Line Interface). Additionally, the development of a Bash CLI is underway.

3.3 Library Tour

The Dataverse interface is designed to be intuitive and user-friendly, substantially simplifying the data processing workflow. Careful consideration has been given to user experience, minimizing the learning curve for new users and enabling them to rapidly understand and effectively utilize Dataverse with minimal effort.

Executing ETL pipeline with configuration. Using Dataverse is straightforward, primarily requiring a properly designed configuration for the execution of the ETL pipeline. The essential configuration elements include specifying the Apache Spark specifications for execution and ordering the data processors to be applied. The initial data processor must be configured for data ingestion to facilitate data loading, followed by any additional data processors the user wishes to employ. We give an example of using Dataverse below, with the configuration simplified for brevity.

```
# import necessary libraries
import OmegaConf
from dataverse.etl import ETLPipeline

# set up configuration
config = OmegaConf.create({
    'spark': {Spark spec},
    'etl': [
        {data ingestion}
        {cleaning}
        {deduplication}
        {data saving}
      ]
  })

# run on ETL pipeline
etl = ETLPipeline()
spark, dataset = etl.run(config)
```

Incorporating custom data processors. Integrating a custom data processor into Dataverse requires defining a custom function and decorating it using `@register_etl`. The custom function requires only two mandatory inputs, a Spark instance and the input data. Thus, creating custom operations in Dataverse is a natural extension for those with proficiency in Spark. An example of adding custom processors is given below.

```
# add your custom process
@register_etl
def add___one___func(spark, x):
    x = x.map(lambda x: x + 1)

    return x

# add to configuration
config = OmegaConf.create({
    'spark': {Spark spec},
    'etl': [
        {data ingestion}
        {add___one___func}
        {cleaning}
        {deduplication}
        {data saving}
      ]
  })

# run on ETL pipeline
etl = ETLPipeline()
spark, dataset = etl.run(config)
```

Scaling with AWS EMR. As explained in the previous section, Dataverse natively supports AWS integration to provide a solution for users facing local resource limitations. To leverage the power of AWS EMR, users can simply add a single argument when running their own ETL pipeline. An example usage is given below.

```
# run on AWS EMR
etl = ETLPipeline()
etl.run(config, emr=True)
```

Debugging with helper functions. To facilitate debugging, Dataverse provides helper functions such as generating fake data. Further, users can start debugging at any point within the pipeline by retaining only the steps up to the point they wish to debug in their own ETL pipeline.

```
config = OmegaConf.create({
    'spark': {Spark spec},
    'etl': [
        {generate_fake_data}
      ]
  })
etl = ETLPipeline()
spark, x = etl.run(config, emr=True)

# start debugging with your output from
    this line
print(x.show())
```

4 Related Work and Background

4.1 Distributed Processing for Massive Datasets

The processing of big data has presented significant challenges since the advent of the internet era. In the early stages of deep learning, models were developed for specific purposes using relatively small datasets. However, the emergence of large language models (LLMs) has necessitated the use of massive datasets, rendering distributed processing an indispensable requirement. Rather than relying on single nodes, multi-node and multi-processing environments enabled by open-source tools such as Slurm (Yoo et al., 2003) and Spark (Zaharia et al., 2016) have become essential. LLM-aware data processing tools have been designed with distributed processing architectures in mind to address the immense computational demands.

4.2 Data Quality Control for Large Language Models

Ensuring data quality at a massive scale presents formidable challenges. Manual inspection of the data is impractical due to its sheer volume. The emphasis on data quality control has become crucial (Penedo et al., 2023; Choi and Park, 2023), primarily because the pursuit of larger datasets often involves incorporating low-quality data that has not undergone meticulous human curation (Li et al., 2024b; Chung et al., 2022). One of the most notable examples of such massive datasets is Common Crawl (Dodge et al., 2021), often regarded as the holy grail of web data. However, this indiscriminately crawled data from the internet frequently suffers from a myriad of issues, including duplicated content, excessive brevity or verbosity, hidden biases, and the inclusion of junk data. To address these challenges, implementing a range of strategies for data quality enhancement is essential, among which deduplication is particularly critical (Lee et al., 2022b). Even when utilizing high-quality datasets, the possibility of encountering duplicated data remains, as multiple sources may be incorporated. Another key strategy could be the elimination of benchmarks or other unintended data inadvertently included in the dataset, which is known as decontamination. Additionally, removing overly short or excessively long sentences could be essential for maintaining data integrity (Moon et al., 2023; Li et al., 2024a).

4.3 ETL (Extract, Transform, Load)

ETL, stands for Extract, Transform, Load, is a fundamental process that involves gathering data from various sources and consolidating it. In the step of "Extract", Dataverse retrieves raw data and prepares it for processing. During "Transform", the data undergoes various processes such as deduplication and cleaning. Finally, Dataverse performs "Load" step which transfers the processed data into a storage destination of choice. Incorporating all these ETL steps enables end-to-end data handling from multi-sources.

5 Conclusion

To address the surging needs of data processing at massive scales, owing to the rise in popularity of LLMs, we propose Dataverse, an open-source library for ETL pipelines designed with scalability and future growth in mind. Dataverse is user-friendly designed with a block-based interface, allowing users to easily add custom functions for data processing while also natively supporting a wide array of commonly used data operations. Furthermore, Dataverse offers scalable solutions through its seamless integration with Spark and AWS EMR, allowing users to use Dataverse to handle data workloads of varying sizes. We envision Dataverse becoming a central hub for LLM data processing, facilitating collaboration, knowledge exchange, and ultimately accelerating advancements in the field.

Limitations

Although the architecture of Dataverse can support multi-modal data such as image or video as well text data, the current implementation of Dataverse has yet to bring such features. However, we plan to incorporate image and video support in future releases to maintain alignment with emerging research trends and evolving demands.

The Spark-based architecture of Dataverse necessitates tuning and optimization by experienced data engineers to achieve peak performance and scalability. While we have implemented reasonable defaults, we acknowledge that the current version may not fully unlock the potential inherent in Spark. For further optimization, we plan to add an automatic configuration feature that reasonably maximizes the Spark performance.

Ethics Statement

We recognize that LLMs can reflect biases present in their training data, potentially generating skewed results across dimensions like race, gender, and age. While Dataverse incorporates bias mitigation techniques, ongoing monitoring and improvement are necessary.

The collection of massive datasets also raises privacy and copyright concerns. Dataverse aims to minimize these risks through anonymization and filtering, but vigilance is still required throughout the data acquisition and processing pipelines.

We are keenly aware of these ethical challenges in developing Dataverse. We are committed to continually enhancing Dataverse's capabilities to address bias, privacy, and potential misuse concerns. Our goal is to provide a powerful tool for advancing language AI while upholding robust ethical principles and mitigating societal risks to the greatest extent possible.

References

Eujeong Choi and Chanjun Park. 2023. Dmops: Data management operation and recipes. *arXiv preprint arXiv:2301.01228.*

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *Preprint*, arXiv:2210.11416.

Jesse Dodge, Maarten Sap, Ana MarasoviÄĞ, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *Preprint*, arXiv:2104.08758.

Tanmay Garg, Sarah Masud, Tharun Suresh, and Tanmoy Chakraborty. 2023. Handling bias in toxic speech detection: A survey. *ACM Computing Surveys*, 55(13s):1–32.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361.*

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022a. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022b. Deduplicating training data makes language models better. *Preprint*, arXiv:2107.06499.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. 2024a. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. *Preprint*, arXiv:2402.00530.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024b. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. *Preprint*, arXiv:2308.12032.

Nicholas D Matsakis and Felix S Klock II. 2014. The rust language. In *ACM SIGAda Ada Letters*, volume 34, pages 103–104. ACM.

Hyeonseok Moon, Chanjun Park, Seonmin Koo, Jungseob Lee, Seungjun Lee, Jaehyung Seo, Sugyeong Eo, Yoonna Jang, Hyunjoong Kim, Hyoung gyu Lee, and Heuiseok Lim. 2023. Doubts on the reliability of parallel corpus filtering. *Expert Systems with Applications*, 233:120962.

Chenghao Mou, Chris Ha, Kenneth Enevoldsen, and Peiyuan Liu. 2023. Chenghaomou/text-dedup: Reference snapshot.

Guilherme Penedo, Alessandro Cappelli, Thomas Wolf, and Mario Sasko. 2024. Datatrove: large scale data processing.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only. *Preprint*, arXiv:2306.01116.

Paul M Schwartz and Daniel J Solove. 2011. The pii problem: Privacy and a new concept of personally identifiable information. *NYUL rev.*, 86:1814.

Seongjin Shin, Sang-Woo Lee, Hwijeen Ahn, Sungdong Kim, HyoungSeok Kim, Boseop Kim, Kyunghyun Cho, Gichang Lee, Woomyoung Park, Jung-Woo Ha, et al. 2022. On the effect of pretraining corpora on in-context learning by a large-scale language model. *arXiv preprint arXiv:2204.13509*.

Robik Shrestha, Kushal Kafle, and Christopher Kanan. 2022. An investigation of critical issues in bias mitigation techniques. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1943–1954.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. 2024. Dolma: An Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*.

Yau-Shian Wang and Yingshan Chang. 2022. Toxicity detection with generative prompt-based inference. *arXiv preprint arXiv:2205.12390*.

Zige Wang, Wanjun Zhong, Yufei Wang, Qi Zhu, Fei Mi, Baojun Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023. Data management for large language models: A survey. *arXiv preprint arXiv:2312.01700*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. 2016. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, 104(9):1681–1710.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*.

Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer.

Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Discussion

Slow Progress in Open-Source Development for LLM Data Processing Notwithstanding the emergence of LLMs in the recent past, a paucity of widely adopted open-source solutions persists within the domain of data processing for these models. The substantial computational costs and significant computing power requirements have predominantly confined advancements to well-funded or large-scale organizations. Consequently, this has rendered LLM research inaccessible for individuals and smaller entities. Initially, the demand for open-source solutions in LLM research was not substantial. However, as smaller LLMs began to emerge, empowering individuals and smaller entities to participate in LLM research, the necessity for open-source solutions became increasingly evident. It is becoming progressively clear that to foster wider accessibility to LLM research and to facilitate greater participation in the field, there is an urgent need to accelerate the development of open-source solutions, with a particular emphasis on data processing.

Quintessential Elements for Developing Open-Source LLM Data Processing Solutions The quintessential elements for successfully developing open-source LLM data processing solutions can be distilled down to three key aspects: a user-friendly interface, cost-efficiency in data processing, and an automatic resource assessment tool.

Firstly, and most crucially, a user-friendly interface is imperative to ensure widespread community adoption. The objective should be to create an interface that is as intuitive as a computer's power button, thereby encouraging heightened user interaction and utilization.

However, a focus on user experience becomes effective only when underpinned by reliable, highly optimized, and cost-effective data processing capabilities. The utilization of data processing tools in the LLM era can be financially onerous. Consequently, these tools necessitate judicious calibration to ensure cost-effectiveness. Users must not encounter repeated attempts due to errors, as this not only adversely impacts the user experience but also exacerbates the associated costs.

Finally, the establishment of a system that automatically assesses the available computing power and evaluates its suitability for the given data is paramount. By doing so, it aims to preclude users from experiencing wasted time and frustration due to insufficient processing capabilities.

Registry-based System for Efficient Data Processor Management One might question the rationale for employing a registry, given that it can introduce complexity, particularly in multi-user systems due to synchronization issues. However, Dataverse operates as a single-user system, thereby eliminating the need for synchronizing the registry among users. This approach resolves the synchronization challenge and confers two key advantages. Firstly, it eliminates the necessity for importing data processor functions using relative paths, thereby simplifying the development process. Secondly, it enables an auto-registration system, alleviating the burden on users of having to manually save the data processing functions within the package. Instead, users are afforded the flexibility to implement their data processing functions in their preferred locations, without being constrained to a specific directory. Consequently, custom functions can be located in various environments, including Jupyter Notebooks, and can be seamlessly integrated into the ETL pipeline.

Block-Based Coding for Enhancing Experimentation The block-based coding approach essentially presents a data processor as a singular block, and an ETL Pipeline as a composite structure of multiple blocks. This design paradigm affords users the flexibility to add, remove, or reshuffle blocks with ease, achievable merely through configuration settings. Consequently, it enables users to effortlessly experiment with limitless combinations without the need to modify the codebase.

Batch Processing: An Enduring Approach in the Context of Large-Scale Data In the context of large-scale data, accuracy takes precedence over speed. The objective is not merely to thoughtlessly utilize the influx of data but rather to focus on producing high-quality and reliable data. To attain this, global assessment must encompass deduplication and ensuring a balanced perspective to avoid bias. This becomes substantially challenging in real-time data processing. As a result, Dataverse still heavily relies on batch processing, as it is designed for LLM data preparation, where accuracy and quality are paramount.

Naming Convention: Rationale Behind the Unconventional ___ Selection Naming large quantities of data processors, as is the case in Dataverse, presents two primary challenges: maintaining uniqueness and ensuring usability. With the potential for adding up to 10,000 data processors, guaranteeing unique identification can be daunting. Hence, the idea of categorization on two levels emerged, which not only ensures uniqueness but also makes the data processors easily identifiable and comprehensible.

However, a discussion emerged regarding whether or not to integrate these categories into the data processor's name. The confusion arises when functions like **remove** appear in multiple categories such as **deduplication** and **cleaning**. How do we clearly identify the difference? This problem was mitigated by asking users to provide the category and name as separate arguments within configurations. However, this proved to be cumbersome, and hence these elements were combined into a single character string. The separator underscores (___) were then introduced to distinctively separate the ETL Category, ETL Sub-Category, and ETL Name. Consequently, the unconventional naming convention of [ETL Category]___[ETL Sub-Category]___[ETL Name] was employed.