

Data-Juicer: A One-Stop Data Processing System for Large Language Models

Daoyuan Chen*, Yilun Huang*, Zhijian Ma*, Hesen Chen*, Xuchen Pan†, Ce Ge†, Dawei Gao†, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li‡, Bolin Ding‡, Jingren Zhou
Alibaba Group

ABSTRACT

The immense evolution in Large Language Models (LLMs) has underscored the importance of massive, heterogeneous, and high-quality data. A data recipe is a mixture of data of different types and from different sources for training an LLM, which has been known as one of the most important factors that decide the LLM’s performance. Existing open-source tools for LLM data processing are mostly tailored for preparing specific data recipes. To continuously uncover the potential of LLMs, incorporate (after cleaning) data from new sources, and improve LLMs’ general-purpose or domain-specific performance, we build a data processing system, named Data-Juicer, with which we can efficiently generate diverse data recipes, explore different possibilities in forming the data mixtures, and evaluate their effects on the model performance. Different from traditional data-analytics pipelines, Data-Juicer faces some unique challenges. Firstly, the possible data sources for forming data recipes are truly heterogeneous and massive with various qualities (e.g., considering all web-pages on the Internet). Secondly, it is extremely expensive to precisely evaluate data recipes’ impact on the LLMs’ performance. Thirdly, sufficient flexibility needs to be provided to the end users of Data-Juicer, model developers, to configure and evaluate different data recipes.

Data-Juicer features a fine-grained abstraction of the pipeline for constructing data recipes, with over 50 built-in operators that can be freely composed and extended. By incorporating visualization and auto-evaluation capabilities, Data-Juicer enables a timely feedback loop after data processing for both LLM pre-training and fine-tuning. Further, Data-Juicer is optimized and integrated with ecosystems for LLM training, evaluation, and distributed computing. With the help of Data-Juicer, we derive data recipes that achieve remarkable performance boosts on state-of-the-art LLMs, demonstrating up to 7.45% increase in averaged score across 16 LLM benchmarks and 17.5% higher win rate in pair-wise GPT-4 evaluations. More importantly, we hope that Data-Juicer promotes broader data-centric research on training and understanding LLMs. Data-Juicer and our data recipes are released and actively maintained at <https://github.com/alibaba/data-juicer>.

1 INTRODUCTION

Large Language Models (LLMs) [9, 18, 69, 70, 90, 92] have achieved unprecedented intelligence, enabling applications that would otherwise be infeasible due to unsatisfied performance. As the “food” for LLMs, *data* plays a pivotal role in these exciting advancements [31, 62, 71, 103]. LLMs are built by pre-training on large-scale

general-purpose corpus and are fine-tuned with specific-purpose data for alignment or downstream tasks. For pre-training data, a collection of diverse data, including web texts, dialogues, academic papers, code bases, and others, help to develop the vast repository of knowledge and great applicability [9, 57, 75]. Fine-tuning data, which further refines LLMs and aligns model behavior with human values [3, 48, 68]. As “garbage in, garbage out” suggests, the input data for training or tuning an LLM has a direct impact on the quality of the derived model [35, 44]. Building effective data processing solutions for LLMs remains a sophisticated yet fully under-explored task, given the common challenges in processing both pre-training and fine-tuning data, which pursue good data quality, proper data diversity, and large data volume.

Unfortunately, there exist only a few open-source projects contributing their LLM training data and the corresponding processing codes [24, 51], particularly in comparison to numerous open-source projects on models and training infrastructures [6, 7, 19, 67, 80, 93, 105]. Such limited development of data processing will obstruct the progress of quantitatively understanding and enhancing LLMs from the perspective of data, especially accompanied by the following noteworthy Challenges for LLM data processing.

(C1) High Heterogeneity in LLM’s Data Recipe. LLMs involve several developmental stages and enable diverse usages including coding and dialog assistance, and even aiming at Artificial General Intelligence. As a result, they demand an extensive variety of data types, formats, and quality in their training data, leading to highly complex data-processing pipelines. A *data recipe* for training or tuning an LLM is such a mixture of processed data from different types of sources, with their ratios and processing pipelines properly set [24, 25]. Existing systems, e.g., [24, 80], release certain processing scripts to generate data recipes for the pre-training purpose, whereas [17, 92] focus on data recipes for improving data diversity and quality in LLaMA’s [93] fine-tuning stage. However, due to the lack of abstraction of processing pipelines and composability of operators (OPs), such as those for data editing, cleaning, and filtering, it is difficult to incorporate new data sources in data recipes provided by these systems, or to extend their pipelines for exploring other possibilities of data recipes.

(C2) Timely Feedback for Data Recipe. The search space of LLM’s data recipes is huge due to the high degree of heterogeneity in data sources and numerous ways to mix them (with proper processing OPs, combinations, and ratios). We want to explore as many data recipes in the search space as possible with timely feedback to uncover the potential of LLMs and improve their performance. However, as the size of an LLM (number of model parameters) is usually billions or even larger, it is super expensive, in terms of both the time and computational resources, to evaluate the impact

*Co-first authors.

†Equal contribution.

‡Corresponding authors, email addresses: {yaliang.li, bolin.ding}@alibaba-inc.com

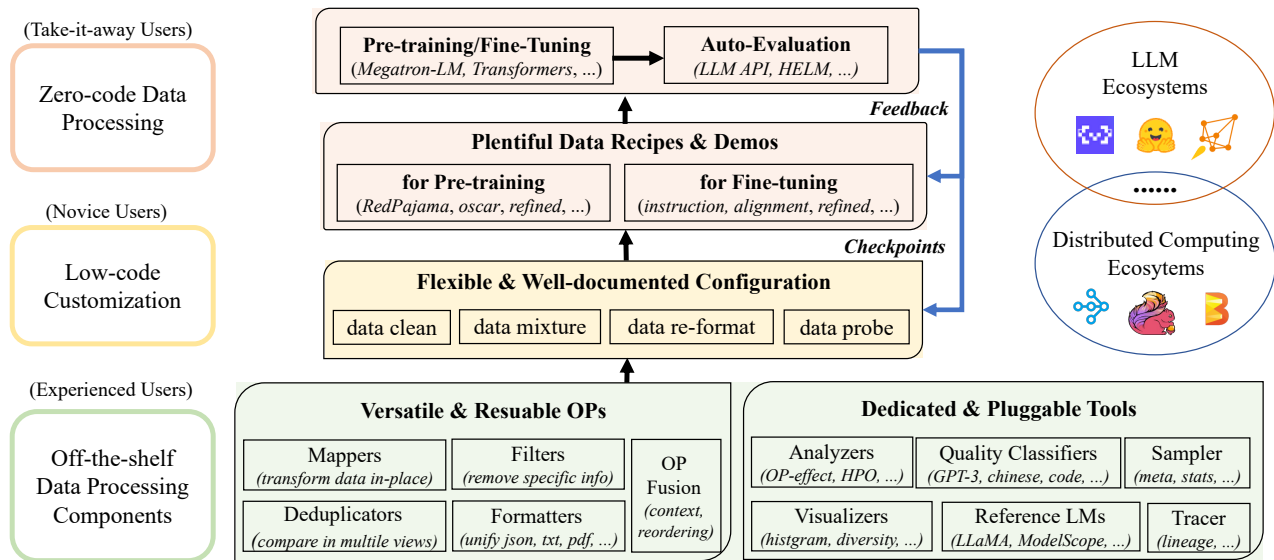


Figure 1: Overview of Data-Juicer.

of a data recipe on the LLM’s performance by training or tuning it with the recipe [85] and running evaluation benchmarks [59].

(C3) Usability and Customizability. The workflow of training or tuning LLMs starts from processing raw data. Exacerbated by the above two challenges, there is an urgent need for a data-centric infrastructure, so that the model developers can easily re-use or implement their own OPs and tools for data processing, configure their processing pipeline, explore various data recipes, and evaluate the resulting LLMs’ performance. We need such a system to accelerate the exploration and understanding of LLMs’ potentials.

(C4) Massive Data Volume. Last but not least, LLMs are trained on vast corpora, with data volumes stretching to an unprecedented magnitude of billions or even trillions of tokens (a modeling unit of text dependent on the used tokenizer [49]). Efficient LLM data processing of such volume is critical but arduous. However, considerations on system performance optimization are often bypassed by existing studies, leaving significant room for enhancement in ensuring the stability of data processing and facilitating the deliveries of processed data and trained weights for LLMs.

Overview of Data-Juicer. In this paper, we advocate for a one-stop data processing system that addresses these challenges, enabling comprehensive, user-friendly, and efficient data processing abilities to facilitate data-centric LLM research and development. The proposed system, named Data-Juicer and illustrated in a bottom-up view in Figure 1, is strategically designed to generate data recipes making data more “juicy” and digestible for LLMs. We decouple the mixture elements of existing solutions for LLM data processing, such as specific data types, auxiliary models, and downstream tasks. As highlighted by the green boxes, Data-Juicer fosters a fine-grained abstraction and implementation of composable modules with over 50 versatile OPs and dedicated tools. We

make Data-Juicer end-to-end configurable to help prepare traceable, comparable, and refinable data recipes at various scenarios of LLM pre-training and fine-tuning, as shown in the yellow and pink boxes. Coupled with established auto-evaluation capabilities, Data-Juicer supports a timely feedback loop at multiple development stages of data recipes and LLMs, thereby promoting the production of valuable LLM data.

To meet diverse user backgrounds and needs (marked by the left three rectangle boxes), we design Data-Juicer as an easy-to-use, flexible and extensible system. Beginners are shielded from underlying complexities and benefit from numerous ready-to-use datasets, data recipes, and pluggable tools, supporting zero-code LLM data processing. With the help of the flexible configuration module, experienced users can simply modify built-in data recipes, reorganize the order of OPs and tools, and tune the value of their hyper-parameters, to meet their lightweight customization needs. Thanks to the standardization and modularization, advanced users are empowered to conveniently extend and register their new OPs and tools into Data-Juicer, facilitating quick engagement in secondary development. Furthermore, we offer more than a dozen interactive tutorials implemented by streamlit [87] to help users with their LLM data processing journey.

Data-Juicer hinges itself on the Huggingface-datasets library [55], providing a unified intermediate representation of data and achieving optimized space-time efficiency and robustness through various techniques such as context management, OP fusion, caching, and checkpoint mechanisms. Furthermore, as the right two circles show, Data-Juicer seamlessly integrates with ecosystems for LLM training and evaluation such as Megatron-LM [85] and HELM [59], and distributed computing ecosystems such as Ray [66] and Beam [5], thus facilitating comprehensive LLM data processing and enhancing large-scale data processing capabilities.

Leveraging the proposed system, we refine several open-sourced datasets and derive numerous data recipes for both LLM pre-trained and fine-tuning. These refined datasets are not only higher in quality but also more digestible by LLMs, leading to effective performance improvements of LLMs. Empirical analysis showcases an improvement of up to 7.45% averaged score across 16 LLM benchmarks using our refined pre-training data. Even pre-trained on only 43% quantity of compared data, we observe superior performance over state-of-the-art (SOTA) LLMs such as Falcon [1]. Moreover, compared with SOTA LLMs fine-tuned on competitive open English and Chinese data, LLMs fine-tuned on Data-Juicer’s data gain an average of 10.3% higher win rate of pair-wise GPT-4 evaluation, while with an average 56.8% fewer data quantity. Finally, we introduce its utility in real-world deployment, and validate its superior system efficiency and scalability of Data-Juicer, by up to 88.7% reduction in single-machine processing time and 77.1% savings in memory usage, and 7.91x distributed processing acceleration.

Contributions. Our contributions are summarized as follows:

- We propose and build a novel system for LLM data processing, Data-Juicer, which is featured by decoupled modules and over 50 versatile OPs and tools. To easily dive into data quality and insights, Data-Juicer fosters a timely feedback loop with interactive visualizations and auto-evaluation capabilities.
- Demonstrated by extensive empirical evidence, Data-Juicer produces numerous high-quality data recipes to enhance LLMs and exhibits superior system performance, powered by dedicated optimization and integrated distributed computing ecosystems.
- We integrate data-centric methodologies for LLM data processing and LLM development with user-centric interface designs, with the aim that Data-Juicer can ease access for diverse users and democratize LLM data processing.
- To promote further research and development, our system, data recipes, and tutorials are maintained and released at <https://github.com/alibaba/data-juicer>, which we hope can help pave the way for next-generation production paradigms of LLM data.

Organization. The subsequent sections describe Data-Juicer in detail. Sec. 2 elaborates on the background and related studies. Sec. 3 outlines our OP pool, as a response to high heterogeneity of LLM data recipes (C1). Sec. 4 delves into our formulation of timely feedback loops for data processing and development of LLMs (C2). Sec. 5 details our repository of data recipes and tools that counteract usability and customization issues (C3). Sec. 6 expounds on the employed system optimization to tackle massive data volume (C4). Sec. 7 focuses on an extensive empirical evaluation for the quality of data recipes, performance and usability of Data-Juicer. Lastly, we draw a summary in Sec. 8.

2 BACKGROUND AND RELATED WORKS

2.1 Large Language Model (LLM) Data

Large Language Models (LLMs). Language modeling is a crucial component for achieving machine intelligence [65, 109]. In the last few years, this field has witnessed remarkable advancements, particularly with the emergence of the pre-training and fine-tuning paradigms, where language models undergo an initial phase of training with a general-purpose corpus before being fine-tuned

with specific-purpose tasks [27, 72]. This procedure has yielded exceptional performance across a spectrum of natural language processing (NLP) tasks [54, 76].

Recently, taking advantage of the highly parallelizable nature of the self-supervised Transformer architecture, the scales of model parameters and training corpus for LLMs have significantly been increased [28, 69]. Meanwhile, LLMs have aroused considerable interest in the potential of artificial general intelligence [10, 11, 30, 38, 43, 99, 108]. While model-centric studies proliferate, how to better process LLM data remains an intricate domain yet to be completely unfurled, whether for pre-training or fine-tuning data.

Pre-training Data. Pre-training serves as the foundation for LLM intelligence. By being trained on large amounts of high-quality data, LLMs can acquire elementary language comprehension and generation capabilities [37]. Aiming to elucidate the link between data and LLMs intuitively, let us consider a typical pre-training objective prevalent among mainstream LLMs. Given a token sequence $[t_1, \dots, t_i, \dots, t_n]$, an LLM θ is trained to maximize the joint probability of the text as follows:

$$\theta_0 = \arg \max_{\theta} \sum_{i=1}^n \log p(t_i | t_{1:i-1}; \theta). \quad (1)$$

This objective is for auto-regressive language modeling and allows the pre-trained θ_0 to predict the probability of the next token by adhering to the inherent sequential ordering of the language [94].

Exploiting this unified yet simple modeling goal, researchers collect a large volume and diverse range of corpus data, which usually contains hundreds of billion tokens or even trillion tokens. After tokenization and pre-training, LLMs have succeeded in stimulating a wide range of advanced capabilities. The LLM pre-training data generally includes various types derived from the web crawlers [26, 71], dialogues or social media [107], book-length formal texts [36, 110], rigorous encyclopedias and academic texts [31, 100], structured coding texts [18, 57], and more texts from financial, medical and legal domains [58, 91, 104]. A challenge is nonetheless posed in the careful processing and formulation of pre-training data to filter noise, redundancy, irrelevance, and potentially toxic [33, 62].

Fine-tuning Data. Numerous studies have underscored that fine-tuning – the process of refining pre-trained LLMs using a smaller, task-specific dataset – can further enhance or unlock additional capabilities of LLMs [40, 53, 97, 98]. Crucially, this process also paves the way for better aligning the behavior of these advanced models with human values and preferences [60, 68].

In this phase, though the data volume decreases exponentially compared to the pre-training phase, the format of fine-tuning data is quite different [73]. Typically, given a textual dataset $\{(x_1, s_1, y_1), \dots, (x_j, s_j, y_j), \dots, (x_m, s_m, y_m)\}$, the goal of fine-tuning is to adjust the pre-trained LLM θ_0 to find θ^* that maximizes the likelihood of the task-oriented response y_j for the user query x_j :

$$\theta^* = \arg \max_{\theta} \sum_{j=1}^m \log p(y_j | x_j, s_j; \theta); \quad \theta \leftarrow \theta_0. \quad (2)$$

Here s_j stands for task-specific instructions, such as “summarize the following text:”, optionally accompanied by a few demonstrative samples for in-context learning [9].

The fine-tuning data can be broadly categorized into two types: *Instruct Fine-Tuning (IFT)* datasets to enhance the instruction-following

abilities of LLMs and are usually adapted from existing NLP benchmarks [4, 61]; and *Chat Fine-Tuning (CFT)* datasets for enhanced dialog ability and human value alignment [70, 92]. There are preliminary explorations emphasizing the importance of data diversity over volume for fine-tuning data [20, 95]. Several studies also indicate that data types representing human values can potentially lead to degraded general performance, a phenomenon known as the “alignment tax” [70]. However, how to more effectively process the fine-tuning data to maximize its usefulness and minimize potential risks remains an open area for further investigation.

The Symbiotic Nature of Pre-training and Fine-tuning Data.

It is worth pointing out the analogous properties shared between these two types of data, which motivate our synergetic approach when bearing quality, diversity, and volume considerations in mind.

Specifically, the quality aspect of the text has been studied extensively in existing literature [62]. Efforts have been made to enhance aspects such as text structure, the soundness of arguments, contextual richness, writing correctness, comprehensiveness, levels of anonymization, and harmlessness. The widespread implementation of cleaning, deduplication, and anonymization processes in pre-training data typifies the aforementioned pursuit. For example, researchers may opt to iterate over additional epochs with Wikipedia-style data in LLM training [93]. Similarly, fine-tuning data processing also employs filtering, deduplication, and detoxification strategies, aiming to enhance the user experience and the degree of aid offered by LLMs [17, 33].

Diversity is another shared property studied at length in both types of data. Mixing various types of data and finding suitable mixture weights to achieve appropriate diversity has been a primary concern in works for pre-training data processing [103]. Analogously, efforts for fine-tuning data aim to increase multi-view diversity such as tuning tasks and expression styles, which further underscores this shared property [70, 77, 92].

In addition, the pursuit of quality and diversity tends to trade off with data volume, which is also reflected in these two types of data. Researchers have incessantly strived to empower LLMs with massive amounts of data, hoping to encapsulate as much human knowledge as possible. For instance, there has been an influx in pre-training data volumes to terabyte levels [69, 71], and fine-tuning data volumes have grown from mere thousands to millions [4, 96]. However, the counter effects of these initiatives are also brought into these large volumes of data, including heightened noise, potential inferior quality, and increased bias, which necessitate additional data processing efforts and surging LLM training overheads.

2.2 Existing LLM Data Processing Solutions

LLM data processing is an early area that is still working towards common standards, and we aim to embody a pioneering system for the community. With a commitment to open-source ethos, Data-Juicer caters to the increasing demand for versatile, flexible, user-friendly and efficient *data processing* solutions, details of which will be described later. This contrasts the well-known LLMs that were largely closed-source in *data* or *data processing*, such as the GPT derivatives [9, 18, 69, 84], LLaMA derivatives [16, 19, 89, 92, 93], and others [1, 15, 79, 102, 107]. While some progress has been made in the open-source LLM data processing landscape [4, 24, 51, 86],

they have not fully delivered the abstraction and breadth of functionalities that Data-Juicer aims to bring to the forefront of the field.

Examining this from the perspective of the target datasets, existing works typically *fixate on specific data sources and use cases for LLMs*, spanning alignment of specialized English sub-datasets for LLaMA pre-training [93], assembly of multi-lingual corpora for pre-training [51], or crowdsourcing for fine-tuning prompt data [4]. However, they lack the systematic and modular processing abilities required to proficiently manage heterogeneous data, which is an area Data-Juicer strives to push its boundaries. These limitations become especially apparent when handling new data types, engaging in language transfer, or implementing particular data cleaning and transformations for LLM applications.

Moreover, existing works suffer from sub-optimal *usability and ability to explore data insight*. Most of these works only offer the processed data along with purpose-built processing codes specific to those data, lacking in ease-of-use considerations and support of assistive tool-kits. This hinders their adaptability to diverse users and alternative usages. Users might face a daunting task when substituting data processing goals or conducting analyses due to a dearth of complementary data-analytical capabilities. The re-development of data processing tools and analytical methodologies, specifically tailored for LLMs, remains largely uncharted territory.

Furthermore, the focus of current works gravitates towards functionality rather than *system performance*, leaving large room for enhancement in efficiency, space management and scalability. Noteworthy shortcomings include reliance on single-machine Python scripts, inappropriate handling of large-scale data, and poor processing speeds due to the utilization of Python’s plain dict object.

We will provide further empirical comparisons in terms of both the quality of the generated data recipes (Sec. 7.1) and the performance of the data processing system (Sec. 7.2).

3 STANDARDIZED OPERATOR POOL

In addressing the heterogeneity of data recipes for LLMs (Challenge 1 in Sec. 1), we devise a set of standardized operator (OP) pool. As outlined in Table 1, the OPs are organized into four primary categories: *Formatters*, *Mappers*, *Filters*, and *Deduplicators*, which incorporate diverse categories, functions, inputs, processing levels, outputs, and application scenarios. Core principles of decoupling and composability guide their structuring, resulting in a varied yet standard set of procedures that contribute to flexibility and user interaction at multiple processing levels. This strategic implementation enhances reusability and reduces complexity, aiding streamlined and decoupled data recipe construction.

3.1 Unified Data Representation

We first introduce *Formatter* OPs designed to unify diverse data sources into an intermediate data representation. Specifically, we choose to build Data-Juicer upon Huggingface-datasets [55] due to its compatibility with mainstream LLM datasets and its column-oriented storage ability backed by Apache Arrow [2]. Our Formatters maintain data objects that are instantiated from several unified base classes that simplify the process design for follow-up OPs and facilitate data accessing efficiency. We support numerous text input

Table 1: Overview of the operator (OP) pool in Data-Juicer, with a detailed list continuously maintained at the official documentation: <https://github.com/alibaba/data-juicer/blob/main/docs/Operators.md>.

| Category | Function | Input | Process Level | Output | OP Usage Examples |
|---------------|---------------------------|--------------------------|------------------------------|-----------------|--|
| Formatters | Data format unifying | Dataset | Dataset | Dataset | Load and unify dataset-hub, txt, json, md, codes, html, pdf, docx, ... |
| Mappers | In-place text editing | Sample | Single-sample; Multi-samples | Sample; Samples | Transform specified headers, textual elements; Fix messy codes; Enable text enhancement |
| Filters | Conditional text removing | Sample | Single-sample; Dataset | Boolean | Filter by meta-info, stats (e.g., lines count); model scores; external resources (e.g., flagged words) |
| Deduplicators | Duplication removing | Single or Paired Dataset | Dataset | Dataset | Compare with hash-based and vector-based deduplication methods |

formats - txt, JSON, parquet, html, md, pdf, code files such as .py and .cpp, amongst others - and homogenize them into a structured format composed of certain columns with *nested access support*, which are conceptually organized by three primary parts “text”, “meta”, and “stats”. These parts respectively hold the raw textual data, metadata information (e.g., date and version), and statistical data that can be generated and consumed by Data-Juicer’s other OPs and tools. This interface works at either the text *sample* or *dataset* level, and is independent of underlying in-memory or disk data layout, alleviating the potential worry over heterogeneous data formats by OP developers.

3.2 Versatile Data Processing

Next, we elaborate on the functionality of the OP pool in Data-Juicer, which is pivotal to the comprehensive data processing tailored for LLMs. Besides the Formatters, which play an essential role in unifying data formats and ensuring a consistent and efficient data flow throughout the processing pipeline, we now give more details about the other three types of data-transformation OPs in Table 1.

Mappers facilitate crucial functionalities of in-place text editing, necessary for single-sample or multi-sample processing across various needs of LLM data processing, such as modifying texts for pre-training and enhancing text diversity for fine-tuning. They effectively handle processing tasks like the removal of specific file headers, messy code rectification, and text enhancements.

Filters come into play by conditionally filtering texts via individual-sample metrics, dataset-level statistics, or external resources like stop-word lists. In doing so, they can eliminate unnecessary text samples, contributing to data focus, cleanliness, and the cost reduction of follow-up LLM training processes significantly.

Deduplicators reduce potential storage waste and improve efficiency. As indicated by several studies [13, 47, 52], duplicate samples adversely affect both the pre-training stability and the performance of LLMs. Besides, Deduplicators help prevent unintentional data leakage during training into evaluation benchmarks, particularly for zero-shot or few-shot tasks [39]. To ensure accurate detection and removal of duplication, we provide efficient and robust methods including hash-based and vector-based comparisons [8, 14, 81].

It is noteworthy that the outputs of Filter OPs are Booleans, which helps to decouple the implementations of actual data processing and computation for various statistics. This dedicated segregation results in two key advantages. Firstly, it enables our dedicated analyzer-related tools (detailed in Sec. 5.2) to utilize these computed statistics for the entire dataset, rather than a filtered subset. Users are also allowed to generate fingerprints for specific partial samples. Secondly, this decoupling enhances compatibility between Huggingface-datasets and Data-Juicer, thereby enabling the efficient reuse of the DATASET.MAP and DATASET.FILTER interfaces to perform these sub-processes in a streamlined manner. As a result, users can effortlessly extend their own custom OPs that only vary from existing OPs in specific partial processing behaviors. In Appendix A.1, we offer an illustrative code example of this decoupling in Listing 1.

3.3 Composability

Data-Juicer’s OPs serve as a testament to our system’s versatility. They enable users to effortlessly process a variety of data types in a composable and modular manner, showcasing Data-Juicer’s dedication to user adaptability and high-quality data production for LLMs. Besides the *functions*, *inputs*, *outputs* and *processing levels* summarized in Table 1, this composability is embedded in more facets, including the *fields to be processed*, *OP hyper-parameters*, and *recommended use cases of each OP*.

Each OP in Data-Juicer is designed to serve a distinct function and can be commanded by users to process different text fields. For example, OP A could process the sample field “text.abstract”, while OP B could focus on “text.main_body”. By default, each OP process on “text” field, which can be freely specified to other “meta” or “stats” related data fields according to users’ needs. This adaptability allows for immense flexibility by simultaneously using OPs with different fields, enabling users to easily manipulate specific text snippets such as removing GitHub codes based on their star counts.

Moreover, these OPs establish a one-size-fits-all solution that encompasses a multitude of configurable parameters such as the number of tokens, filtering thresholds, auxiliary models, and much more. This adjustability of a single OP, amalgamated with the composability of OP pipelines, empowers Data-Juicer to manage a spectrum of input, output, and processing granularity, contributing to its powerful processing abilities.

For usage combinations, OPs are labeled with typical usage scenarios. We maintain OP tags as general usage, LaTeX source files, programming codes, financial data processing, or language-specific processing such as English and Chinese, and so on. These labels facilitate easy navigation and operation, underscoring our aim to blend simplicity with power in Data-Juicer’s architecture.

4 FEEDBACK-DRIVEN DATA PROCESSING

Addressing Challenge 2 outlined in Sec. 1, we incorporate a dynamic feedback loop into the data processing pipeline, which allows users to process and understand data effectively via built-in visualization and automated tracking abilities. As demonstrated in Figure 2, our system (Data-Juicer) enables timely perception and swift iterative refinement of data recipes (indicated by the left and upward arrows) within a holistic feedback loop of LLM data processing and LLM training (indicated by the right arrows).

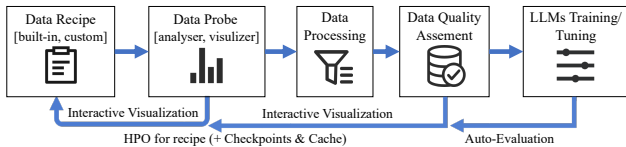


Figure 2: The feedback loop of Data-Juicer.

We will discuss the modeling of the data processing feedback in a hyper-parameter optimization (HPO) perspective (Sec. 4.1), and go through the utility of the interactive visualization (Sec. 4.2), and the integration of ecosystems for LLM training and evaluations (Sec. 4.3). The synergy of these techniques offers an efficient and effective solution to debug and dive into LLM data processing.

4.1 HPO for Data Processing

In Data-Juicer, we incorporate the concept of hyper-parameter optimization (HPO) into the data processing procedure. This is done by tying data-processing-specific hyper-parameters to a variety of feedback signals, including custom target metrics and visualization results. We enhance our system’s functionality by innovatively speeding up the data processing iteration through Checkpoint and Caching mechanisms, and by integrating an automated HPO tool.

4.1.1 Acceleration with Checkpoint and Caching. LLM data processing often necessitates frequent re-conduction due to the alterations in OP hyper-parameters and potential conduction failures, such as exceeding available memory, disk or pre-defined time limits, especially for massive datasets. Accordingly, we provide built-in checkpoint and caching management to foster resilient and reliable data processing. Based on a carefully organized directory structure, Data-Juicer automatically monitors every running process for configuration changes, and creates new files to safely store data and processing states only when any error or exception occurs. While the checkpoint preserves the whole dataset and processing state enabling complete recovery of the processing site, the cache solely saves the dataset object for each OP and is more suited for smaller-scale adjustments as it reduces the overhead of pre-order caches. These techniques allow for a swift recovery during system restarts

or failures, reverting to the most optimal recent processing state stored in the checkpoints, thus mitigating processing redundancy and increasing the feedback frequencies.

Additionally, the proposed state-saving mechanism enables a flexible space-time trade-off at different feedback stages. Users have the option to save states after each OP in the data processing flow, ensuring minimal re-execution time at the cost of maximum storage overhead. Conversely, they could choose to only save the last OP’s checkpoint and cache, incurring minimal storage overhead but increased re-execution time, especially when needing to revert to early steps in the process.

To facilitate a good space-time trade-off, we further perform space complexity analysis for individual OPs, which aids in predicting peak space occupancy and guides us in determining how many checkpoints and caches to store based on available space. By default, Data-Juicer actively monitors disk space usage at the start of data processing, and automatically determines if, and when, checkpoints and cache should be deployed. User-specified saving frequencies and rules are also supported. Consequently, strategic checkpoint and cache management reinforce both the resilience and efficiency of the feedback loop for LLM data processing. The detailed space usage analysis can be found in Appendix A.2.

4.1.2 Auto-HPO. We incorporate an automated HPO tool¹ into Data-Juicer to streamline the finding of good data processing hyper-parameters. To reduce search costs of different data recipes, we support leveraging advanced HPO algorithms such as Bayesian optimization [82], progressive early-stop strategies, such as the Hyperband algorithm [56], and built-in LLM-oriented sampling strategies (detailed later in Sec. 5.2). Specifically, given a pre-defined target metric and search space of data recipes, users can conveniently explore the impact of specific data processing hyper-parameters. Here, we give an illustrative example as follows:

Example of Data Mixing with HPO:

Suppose we aim to find a good set of sampling weights for M datasets to be mixed, where our search space is defined as $w_i \in [0, 1], i \in [1, M]$. The pipeline can be structured as follows:

- (1) We specify the target text fields across all M datasets, and unify their meta-tags and name of text fields via Formatter OPs.
- (2) We leverage meta-tag Filters to cater to specific usage scenarios. Here we only include samples with the language tag “EN”.
- (3) A datasets \mathcal{D}_{mix} is generated from the M datasets, with mixture weights $[w_i]$ drawn by the HPO scheduler to maximize the target metric in step (5).
- (4) A pre-configured data processing including de-duplication OPs is executed on the mixed dataset, ensuring dataset cleanness.
- (5) The target metric is calculated on \mathcal{D}_{mix} as $(n/N + s)$, where N is the total number of tokens of all M datasets, n and s is the number of tokens and average quality score of \mathcal{D}_{mix} using built-in GPT-3 quality classifier (detailed in Sec. 5.2) respectively.

The mixture dataset \mathcal{D}_{mix} is iteratively refined by carrying out iterations steps (3)~(5) to get a larger quantity and better quality. □

The HPO results offer a powerful means of visualizing and understanding data insights as shown in Figure 3, where the importance,

¹W&B Sweeps, <https://docs.wandb.ai/guides/sweeps>

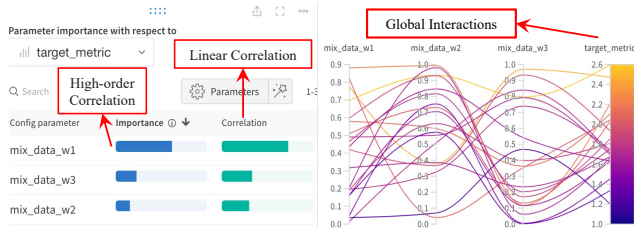


Figure 3: Demonstration of HPO for data recipe.

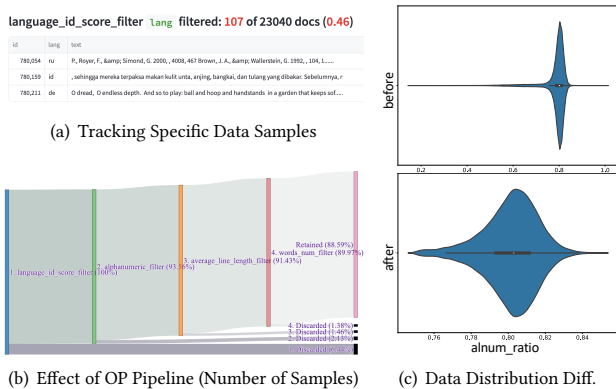


Figure 4: The illustration of interactive visualization of Data-Juicer. More demos are publicly available.

correlation and interaction of w_i for the quality score are estimated and plotted. Besides the quality score demonstrated in this example, the target metric can be customized to include other trade-off terms composed of intrinsic data measures – such as toxicity, helpfulness, or other scores predicted by auxiliary models – or even performance measures of LLMs, such as training loss or benchmark scores (which we will discuss later in Sec. 4.3).

4.2 Interactive Visualization

The ability of interactive visualization is integral to multiple feedback stages of Data-Juicer. Specifically, as Figure 4.(a) demonstrates, users can visually track the effects of individual OPs in terms of the processed data samples. This is facilitated by an innovative built-in tool, **tracer**, which records sample changes after applying each operation for Data-Juicer. For example, tracer presents discarded samples for Filters, pre- and post-editing differences for Mappers, and (near-) duplicate sample pairs for Deduplicators. Coupling this tracking ability with fruitful built-in sampling and visualization tools, Data-Juicer enhances users’ control over the data processing and boosts their confidence and rationals of the process.

Transitioning to the mid-term stage of LLM data processing, Data-Juicer offers a comparative visualization of the data before and after the entire processing from the view of OP pipeline and statistical analysis, as Figures 4.(b) and 4.(c) show. Aided by a built-in tool, **analyzer**, Data-Juicer provides statistical analysis (counts, means, standard deviations, min/max, quantiles, entropy, etc.) to

allow a deep understanding of the data. By default, the summary of per-sample statistics covers 13 dimensions and automatically displays histograms and box plots for each statistical variable, including diverse criteria like sample perplexity, word count, flagged word percentage, and paragraph length, among others. Users also have the flexibility to adjust the dimensions to observe, with a bespoke visualization and data processing experience.

4.3 Feedback with Integrated LLM Libraries

In the later stages of our pipeline, we utilize robust ecosystems designed for LLM training and evaluation, ensuring seamless integration with widely-used libraries such as Megatron-LM [85], DeepSpeed [78], and HuggingFace’s Transformers [101]. With this integration, users can easily train LLMs on datasets produced by Data-Juicer and evaluate their performance to obtain feedback using our pre-built tools and scripts, without getting bogged down in complicated LLM training and evaluation details.

Notably, our system facilitates the timely assessment of model abilities by incorporating multiple dimensions. The system’s capability to swiftly identify potentially ineffective data and training allows us to terminate unwanted LLM data processing promptly. Instead of solely relying on model loss as the basis for evaluating model performance, we support the LLM assessment across various metrics or benchmarks, and track shifts in target scores. Consequently, we can determine whether continued training of an LLM on the produced dataset is justified, thereby helping us minimize data processing and LLM training costs.

Specifically, Data-Juicer’s evaluator supports SOTA LLM benchmarks such as HELM [59], LM-harness [32] and GPT-API-based evaluation [19], as well as the extension of customized evaluation benchmarks and tasks. For a balanced and straightforward evaluation, Data-Juicer supports a leaderboard-style comparison by consolidating results from different target evaluation scenarios, such as ranking averaging, score-normalized averaging, or other customized strategies. The leaderboard-style scoring utility enhances the visualization of strengths and weaknesses of models, guiding subsequent iterations of data recipes and LLMs’ refinements. We also make available *Reference Models* - these are model checkpoints binding with traceable training data in Data-Juicer, popular LLM architectures, training parameters, computation costs, and corresponding evaluation results. They facilitate effortless comparison among different training configurations, particularly for further research on diverse, iteratively developed data recipes.

4.4 Feedback Loop Showcase

The general feedback loop has been discussed before in Figure 2. We now further expound on this by presenting a concrete development example. Here, we intertwine several previously mentioned tools to demonstrate the Data-in-the-LLMdev-Loop process, which results in improved LLM data. As illustrated in Figure 5, we begin with a raw dataset and aim to refine it for better pre-training or fine-tuning of an LLM. The entire process flows as per the following steps:

(1) **Analyze the original dataset.** We can opt to utilize an existing data recipe (a specific configuration file) or craft a new one based on prior understandings of data processing needs. Our built-in Analyzer and Visualizer facilitate this process by computing

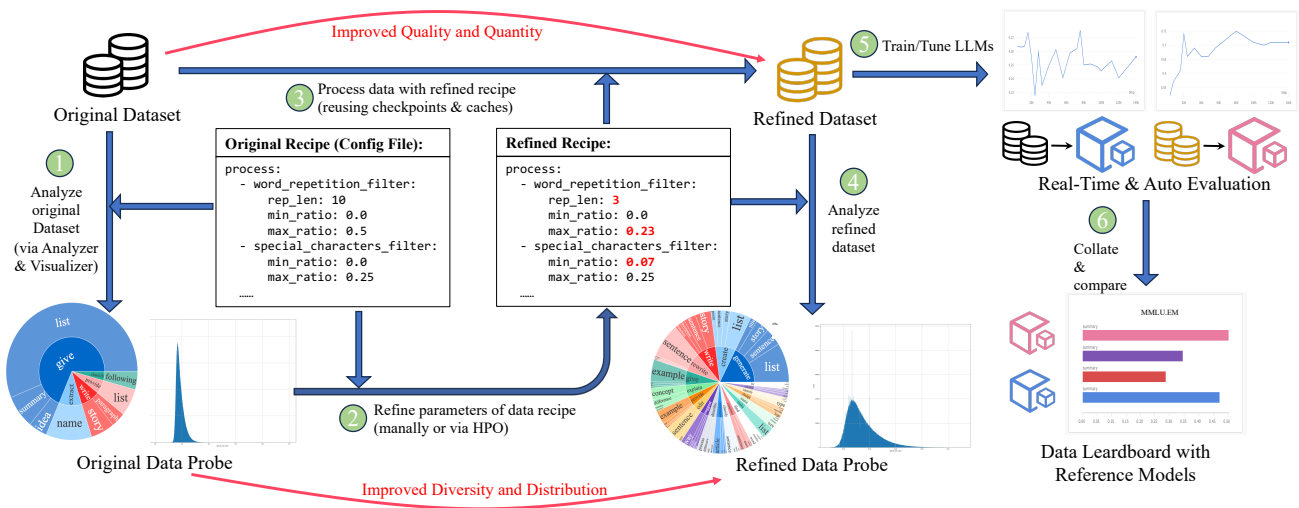


Figure 5: The demonstration of data processing feedback of Data-Juicer, which helps to generate better data recipes for LLMs.

more than a dozen measures such as linguistic diversity, textual statistics, and others to generate a data probe. The two pie plots within Figure 5 indicate the top 20 most common root verbs (inner circle) and their top 4 direct noun objects (outer circle) for the data in field *text.instructions*.

(2) **Refine parameters of the original recipe.** Based on the data probe, users figure out the weaknesses of the original dataset, such as low diversity in expression manners, and long-tail statistics of word counts. Then we refine the parameters in the recipe by adding/removing some OPs or tightening/relaxing filter ranges. During refining, we could find out the effect of each OP easily based on the interactive visualization tool mentioned in Sec. 4.2.

(3) **Process the original dataset with the refined recipe.** Then we process the original dataset with the refined recipe using Data-Juicer and get a refined dataset and several saved checkpoints for further adjustments. This step can be facilitated with the help of our cache and checkpoint mechanisms.

(4) **Analyze the refined dataset.** Like step (1), we analyze the refined dataset again to obtain a new data probe. Based on the statistics and visualization results, we assess the degree of improvement in the data quality. If the refined data fails to meet our expectations, we revert to step 2 to manually adjust the data recipe or employ our HPO tool for automatic refinement (refer Sec. 4.1).

(5) **Get LLMs with the refined dataset.** Then we can train or fine-tune LLMs with the refined dataset and training frameworks integrated into Data-Juicer (Sec. 4.3). During the training or fine-tuning process, our auto-evaluation tools offer timely, multi-view assessments of LLMs. These tools inspect numerous metrics across multiple evaluation datasets. This feature provides us the advantage of halting the process prematurely if the refined data weakens LLM performance, thereby preventing unnecessary costs.

(6) **Collate results and compare with reference models.** Finally, Data-Juicer automatically collates the evaluation results and compares them with reference models in the data leaderboard, providing a clear representation of the effects of data processing alone. Consequently, we derive either a superior LLM, which can be

auto-registered as a reference model, or additional refining guidance from the LLM perspective to further enhance data recipes.

5 BOOSTING USABILITY WITH BUILT-INS

In response to the challenge of varied user customized preferences and technical expertise (Challenge 3 in Sec. 1), we offer an easy-to-use configuration paradigm for data recipes, ready-to-use data recipe templates, and extensive tools, as detailed below.

5.1 Configuring Your Data Recipe

Notably, we make the end-to-end pipeline of data processing configurable in Data-Juicer, including specified processing environment parameters, OP lists, tools used, and so on. This principle of all-in-one configuration ensures reproducibility and traceability, and simplifies changing specifications in data processing, thereby facilitating the formation of data recipes for further refinement and reuse, and enabling the quantitative exploration and automatic optimization of data processing (Sec. 4.1).

Specifically, built upon `Jsonargparse` [46], we provide unified, flexible, easy-to-use and powerful configuration capabilities. It is engineered to automatically register configuration items for OPs and tools, and accept varying sources of configurations such as command line entries, `yaml` and `jsonnet` files, environment variables, default hard-coded values, and a mixture of those for convenient incremental modifications.

For example, users can easily build up their own config files by two recommended methodologies—“subtraction” or “addition”. The “subtraction” approach utilizes a pre-set configuration file containing **all available OPs, tools, and their default parameters**. Users can simply remove or re-order these OPs and adjust these parameters per their requirements. Conversely, the “addition” approach lets users build their configuration files from scratch, leveraging our extensive examples of pre-built data processing recipes for totally

more than 20 high-quality and diverse data recipes for pre-training, fine-tuning, English, Chinese, etc. More quantitative analysis on certain recipes are in our experiments (Sec. 7.1).

5.2 Dedicated Pluggable Tools

To further enhance usability, facilitate system customization and augment users’ data handling capabilities, Data-Juicer includes an extensible collection of powerful dedicated tools that can be conveniently plugged into different stages of the LLM data processing.

Quality Classifier. As an illustrative example, we describe our text quality classifier for culling high-quality text from heterogeneous data sources like CommonCrawl. This tool is a reproduced model based on the closed-source GPT-3 quality scorer [9]. Moreover, we have expanded its applicability to Chinese text and various code types. Encapsulated as a callable pipeline, this tool provides users with the freedom to adapt it to other various scenarios.

The functionality of the classifier is backed by PySpark’s standard Tokenizer or Sentencepiece model [50], along with HashingTF as the feature extractor. It then applies a binary logistic regression classifier to gauge the quality of a text. We provide more empirical verification of them in Sec. 7.2.3.

Enhanced Sampler for LLM data. In Data-Juicer, we have designed several advanced data sampling utilities specialized for large-scale data chunk handling in LLMs. Our solutions effectively streamline representative extraction, optimize processing time and resources, and meet the distinctive needs of LLM developers.

Our stratified sampling technique is noteworthy in this LLM data context. It capitalizes on information within the metadata or statistical fields, thus accommodating varied selection metrics in crafting an effective data sample. To ensure a comprehensive yet flexible representation of the data corpus, we consider various heterogeneous criteria such as document length, token count, the frequency of boolean predicates for post-conditional checks, and even linguistic diversity formulated via occurrences of verb-noun pair (as shown in the pie plots in Figure 2). These dynamic criteria are tailored to distinct analytic needs and promote efficient data processing, seamlessly integrating with downstream OPs and tools.

Full Toolkit. As for other tools, readers can refer to Sec. 4 for an examination of multiple previously discussed tools, including **tracer** and **analyzer** (Sec. 4.2), and **evaluator** and **reference models** (Sec. 4.3). We diligently maintain and evolve the toolkit in Data-Juicer, and make the full set publicly accessible.

5.3 User-Friendly Experiences in Data-Juicer

Data-Juicer is designed not just for functionality but also for adaptability, catering to an extensive user base with diverse expertise and skill sets. While abstracting the intricate system internals, we provide user-friendly interfaces and extensive customizable components. Accordingly, users can embark on zero-code data processing, engage in low-code customization, or delve into in-depth extensions for complex requirements.

- **Zero-Code Processing:** For novice users, Data-Juicer supplies a series of ready-to-use data recipes and plug-in tools for immediate use. This requires no knowledge of advanced system architectures or OPs, as discussed in Sec. 5.1 and Sec. 5.2.

- **Low-Code Customization:** Intermediate users can enjoy the flexibility to alter configurations, data, and external resources to suit their specific needs. They can readily reuse, combine, and edit built-in data configurations; customize quality classifiers and tokenizers; refine data based on our pre-developed recipes; or provide fresh links to auxiliary models or vocabularies from our unified, routinely updated public cloud drive.
- **Advanced Extension:** Experienced users are allowed to easily introduce new OPs by deriving from base classes and implementing their specific “process()” and “compute_stats()” functions, as demonstrated in the code Listing 1. This grants the users an end-to-end view of the process for a single sample, while Data-Juicer handles the nitty-gritty of configuration registration and efficiency optimization.

Additionally, Data-Juicer’s decoupled design facilitates the smooth incorporation of new tools for users at all stages of LLM data processing, ranging from novel visualization dimensions and evaluation datasets to pre- or post-processing scripts.

To enhance the ease of adoption and use of Data-Juicer, apart from the numerous pre-built data recipes (refer Sec. 5), we also provide a series of interactive demos, implemented in Streamlit, for varied profiles and scenarios. This hands-on learning approach has been designed to enable users of varying skill levels to quickly familiarize themselves with and effectively use Data-Juicer.

6 COMPREHENSIVE SYSTEM OPTIMIZATION

To handle large-scale data (Challenge 4 in Sec. 1), we employ a series of optimizations in Data-Juicer from various aspects.

Optimized Computation: Context management, Operator (OP) Fusion and Reordering. To elevate computational efficiency in LLM data processing, we provide advanced context management, operator fusion, and operator reordering techniques for nuanced implementation contributions. The manager meticulously handles shared intermediate variables, such as segmented words, split lines, and others derived from the original textual corpus, across different operators. It allows seamless reuse of these context variables across multiple operators, thereby mitigating the necessity for computationally expensive re-evaluations.

Based on the context manager, the proposed operator fusion method is another new contribution to the field. We propose to identify fusible operators that either share the same contexts or computation sub-procedures. It detects the OP groups first. Successive OPs in the same group should be commutative with each other. It then amalgamates identified fusible operators in each group into a single fused OP, enabling them to be executed faster with a larger localized perspective. The contexts of each sample will be cleaned up after each fused OP, hence little extra memory is required for context management and operator fusion.

Due to the time-consuming increase of single fused OP, we further design a strategy of operator reordering to optimize the execution sequence of the OP list after fusion. For example, based on the commutativity of Filters, we delay the running of time-consuming OPs (such as fused Filters) and prioritize other less time-consuming OPs. As a result, these time-consuming OPs only need to handle fewer samples because the preceding operators have filtered out some of them, enhancing overall computational efficiency.

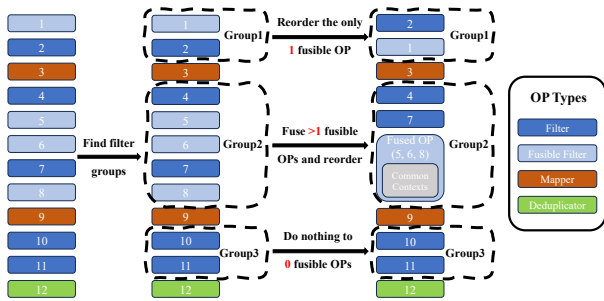


Figure 6: The OP fusion procedure for an OP list.

The whole procedure of OP fusion is summarized in Figure 6. These amalgamation strategies serve dual purposes. Firstly, it minimizes redundant computation, eliminating the need for repetitive yet shared computations. Secondly, it mitigates the overhead of initializing multiple processes by reducing the total count of processing OPs, thus maintaining expeditious data processing routines.

Optimized Space Utilization: Caching OPs and Compression. Recognizing the inadequacies of the original cache management protocol in the Huggingface-datasets library, especially pertaining to the handling of non-serializable third-party models and functions in certain OPs, we design a dedicated hashing method to bypass the serialization procedures of those non-serializable objects, which ensures successful caching of each OP and permits Data-Juicer to leverage optimal cache management.

Furthermore, we incorporated the ability for users to activate advanced compression technologies, such as Zstandard (zstd) [23] and LZ4 [64], in Data-Juicer. It will automatically compress cache files after each OP and decompress these compressed files back to normal cache files when rerunning this OP in the same configuration. Compared with the processing time, compressing/decompressing time is relatively negligible due to the high efficiency of the compression technologies mentioned above. This feature substantially reduces the volume of cache data storage, facilitating the processing of larger datasets without compromising speed or stability.

Optimized Scalability: Distributed Data Processing. The volume of LLM training data can be extremely large, making it difficult to be processed with a single machine. Data-Juicer meshes with distributed processing frameworks such as Ray [66], Apache Beam [5] and Apache Flink [12], and offers the ability to seamlessly translate a data processing pipeline running on a single node into a multi-node cluster. In this way, resources in cluster computing can be utilized to accelerate data processing and generation.

Specifically, we adapt the underlying interfaces of HuggingFace-datasets for those of Ray-datasets, such that all OPs of Data-Juicer, even when written as single-machine Python functions, can be executed in a distributed mode with the help of automatic data partitioning by Ray. An alternative approach we support is to replace the default Ray runner of Data-Juicer with other distributed processing back-ends such as Flink, via pre-translations from Data-Juicer’s processing pipelines into the Beam-compatible ones. As a result, almost all the OPs within Data-Juicer (Mapper,

Filter, and Deduplicator) can be accelerated in a multi-node cluster, and effectively alleviate the bottlenecks on a single node (even with process-based parallelism) caused by memory capacity and IO throughput. More empirical results can be found in Sec. 7.2.4.

In a nutshell, all of these optimizations enhance Data-Juicer’s scalability from various views, to handle the vast amount of data involved in LLMs, ensuring robust and efficient processing while minimizing resource requirements.

7 EVALUATION OF DATA-JUICER

7.1 Making Better Data Recipes

The value of an effective LLM data processing system is reflected not only in its comprehensive and flexible operability but also in its capacity to produce high-quality data that LLMs can more readily “digest”. Data-Juicer provides specialized features for exploring and making data recipes tailored to LLMs, and we have developed numerous ready-to-use data recipes using Data-Juicer. In this section, we evaluate the quality of data recipes generated by Data-Juicer for both LLM pre-training and fine-tuning.

7.1.1 Refined Pre-training Data Recipes. The pre-training data we produced consists solely of publicly available sources, exemplifying the core principles of transparency and reproducibility. Specifically, we choose to improve two widely-used, high-quality datasets for LLMs, TogetherAI’s RedPajama [24] and EleutherAI’s Pile [31], which were curated from 15 highly diverse text sources and subjected to meticulous pre-processing and cleaning to ensure their quality. With the help of Data-Juicer, we further refine them via data analysis, merging and quality enhancement, employing dozens of OPs with varied configurations. For detailed statistics, processing steps and refined data recipes, please refer to Appendix B.2.

To verify the quality of the data recipes derived by Data-Juicer, we use the original RedPajama and Pile, and our refined datasets to pre-train LLMs with mainstream LLaMA architecture and assess the models’ performance across 16 core HELM tasks. We keep the training configurations the same while only modifying the training data. Detailed hyper-parameters are in Appendix B.3.1. The results of average scores of 16 tasks are visualized in Figure 7, where we evaluated checkpoints throughout the pre-training process with an increasing number of billion-sized tokens at 50B, 100B, and 150B. Notably, through fair comparisons with equivalent training tokens, LLMs pre-trained on Data-Juicer-recipes consistently outperformed those using only RedPajama or its union with the Pile, reinforcing the usefulness and effectiveness of Data-Juicer.

Moreover, we compare Data-Juicer-models with several SOTA baselines and summarize the results in Table 2. With only half the data volume (150B tokens), LLaMA-1.3B pre-trained on Data-Juicer-recipe outperformed Pythia-1.4B [6] (300B tokens), and even beats highly competitive Falcon-1.3B [71] trained on 350B tokens. Notably, we further labeled 17 subsets from Alpaca-CoT (a collection of 39 public fine-tuning datasets) with the “Instruct Fine-Tuning (IFT)” tag and performed data mixing and processing using Data-Juicer. Following the usual practice [105], we incorporate these large-volume IFT data into the pre-training phase and execute continuous

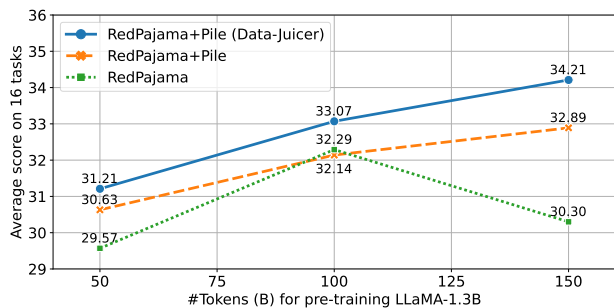


Figure 7: Evaluation results of reference models trained with different datasets but the same pre-training procedures. Data-Juicer’s data recipe gains consistent improvements over baselines.

training upon the checkpoint of Data-Juicer (RedPajama+Pile)-150B. As reflected in the last two rows of Table 2, Data-Juicer gains a further 4.9% relative improvement over the original Alpaca-CoT-IFT while utilizing only ~30% data volume.

Table 2: The average score of the pre-trained LLMs on the 16 HELM core tasks. Individual task results and data recipes are detailed in Appendix B.4. “IFT” denotes the datasets tagged with “Instruct Fine-Tuning” in our context.

| Model | Training Data | #Tokens | Score |
|------------------|-------------------------------------|-------------|--------------|
| Falcon-1.3B [41] | RefinedWeb | 350B | 33.97 |
| Pythia-1.4B [29] | Pile | 300B | 33.96 |
| LLaMA-1.3B | Data-Juicer (RedPajama+Pile) | 150B | 34.21 |
| | + Alpaca-CoT-IFT | 150B + 15B | 35.04 |
| | + Our Refined IFT | 150B + 4.7B | 36.76 |

Taken together, these findings underscore the potential of the Data-Juicer system to generate high-quality data and verify the excellence of Data-Juicer-recipes in terms of enhancing LLM performance while reducing LLM training costs.

7.1.2 Refined Fine-tuning Data Recipes. For the Alpaca-CoT collection, besides the “IFT” tag as validated in Table 2, we also labeled datasets within it with “Chat Fine-Tuning (CFT)” for enhanced dialog ability and aligned human value. To examine their quality, we first use the CFT and EN tags to filter out several competitive subsets, and then generate two new equal-size datasets by random sampling and our designed recipe respectively. Then we conduct fine-tuning on the generated datasets based on the open-source mainstream architecture, English LLaMA-7B [34]. Similarly, we replace the tag “EN” with “ZH”, and use a SOTA LLaMA-2-7B variant [42] for the Chinese scenario. Statistics of these datasets and training hyper-parameters are in Appendix B.3.2.

For a thorough and comparative performance evaluation, we used GPT-4 API for pairwise scoring and tallying of wins and ties.

Table 3: Results of pair-wise model comparisons using GPT4 scoring. “CFT”, “EN” and “ZH” indicate meta-tags as Chat Fine-Tuning, English, and Chinese text respectively.

| Model | Tuning Data | #Samples | Win | Tie |
|-------------------------------------|--------------------|----------|-----------|-----|
| LLaMA-7B [34] | Alpaca | 52k | 16 | 100 |
| | Data-Juicer | 40k | 44 | |
| | Random (CFT, EN) | 40k | 19 | 105 |
| LLaMA2-7B (Chinese, FlagAlpha [42]) | Belle | 543k | 28 | 99 |
| | Data-Juicer | 52k | 33 | |
| | Random (CFT, ZH) | 52k | 19 | 96 |
| | Data-Juicer | 52k | 45 | |

The results are consolidated in Table 3, from which we can see that LLMs utilizing Data-Juicer-recipes consistently demonstrate high validity. Firstly, compared to LLMs trained on the competitive fine-tuning open datasets, Alpaca [92] and Belle [45], LLMs trained on Data-Juicer data gain higher win rates (up to 17.5% for English case) while using less data (up to 90.4% reduction for Chinese case). Secondly, compared to the LLMs trained on the datasets with trivial processing strategy (mixture by random sampling), LLMs trained on Data-Juicer still gain higher win rates (up to 14.4%), which attests to the effectiveness of our enhanced sampling strategy and quality of Data-Juicer-recipes for LLMs again.

7.2 Processing Data Efficiently and Effectively

7.2.1 End-to-End System Performance. To evaluate the processing performance of Data-Juicer, we compare it with two SOTA baselines: TogetherAI’s RedPajama [24] and AllenAI’s Dolma [86]. A more detailed introduction to and comparison with these baselines can be found in Appendix B.3.4. For a fair comparison, here we use their official code repositories and run Data-Juicer on the data recipes with the same OPs to process the Books, arXiv, and C4 datasets, which vary in terms of data sizes, distributions and involve diverse processing OPs.

We conduct multiple rounds of experiments on different numbers of processes (np=[32, 64, 128]) and monitor several core metrics, including processing time and average memory usage. The monitored time is the wall-clock time of the whole processing pipeline. The average memory usage is monitored every second and aggregated across all relevant processes. For more experimental details, please refer to Appendix B.3.3.

The experimental results are summarized in Figure 8. Notably, for all datasets and various numbers of processes, Data-Juicer requires an average of 50.6% less processing time and 55.1% less memory. In particular, it saves at most 88.7% processing time for the arXiv dataset compared with the baseline. Also, it takes up to only 22.9% memory of baseline for Data-Juicer to process the Books dataset, which is mainly because the processing procedure of the baseline loads the whole dataset at once. Overall, Data-Juicer

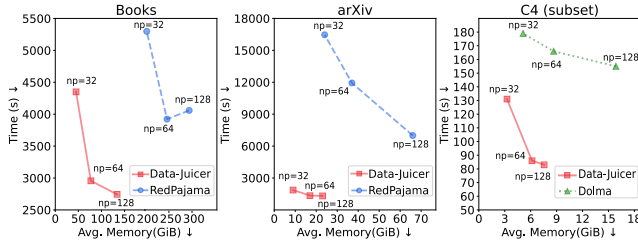


Figure 8: Comparison of stand-alone performance in various data sizes and processing configurations.

effectively alleviates the bottleneck caused by IO of cache files, and achieves better end-to-end time-space efficiency than baselines.

7.2.2 Effect of Context Management, OP Fusion, and Re-ordering. As introduced in Sec. 6, Data-Juicer employs dedicated optimization to minimize redundant computations and save processing time. To examine the optimization effect, we prepared three test datasets of varied sizes and sample counts. Each dataset goes through the same processing recipe which includes 14 OPs (5 Mappers, 8 Filters, and 1 Deduplicator), with 5 of these OPs being fuse-able. We conduct comparison experiments with 4 processes, except for the largest dataset, where we utilize 50 processes to assess if these techniques remain effective on larger scales.

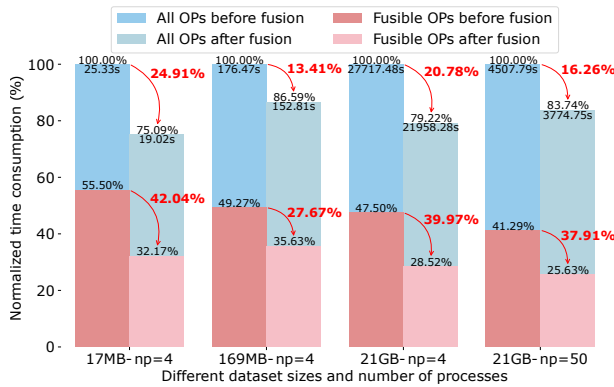


Figure 9: Time comparison before and after OP fusion.

The results are shown in Figure 9, where both the normalized and actual time consumption for each experimental setup are indicated. The results signify that our optimization strategy effectively saves up to 24.91% of the total time for the entire process and saves at most 42.04% of time for those fusible OPs. In addition, the findings showcase that the optimization performs efficiently regardless of variations in dataset sizes or the number of processes utilized.

7.2.3 Effect of Quality Classifiers. As described in Section 5.2, Data-Juicer provides built-in quality classifiers for LLM data processing, and here we present several empirical results regarding their performance. Specifically, we follow the training procedure of the proprietary quality classifier used in GPT-3 [9] and extend its

training pipeline to include Chinese text. In the evaluation of the collected data, we found that our reimplementation of the GPT-3 classifier and its Chinese adaptation achieved F1 scores of 97.47% and 98.64%, respectively. Further training and evaluation details are provided in the Appendix B.1.

Table 4: Comparison of keeping ratio on CommonCrawl.

| Quality Classifier | Keeping Ratio @ label | Keeping Ratio @ Pareto |
|--------------------|-----------------------|------------------------|
| Original GPT-3 | - | 1.30% |
| OUR GPT-3 | 3.22% | 1.41% |
| CHINESE | 1.81% | - |

Furthermore, we assess the filtering effectiveness of these classifiers by comparing their keeping ratios on CommonCrawl. The results are summarized in Table 4, where we employ two data keeping methods used in GPT-3: (1) **label**: $doc_score > 0.5$; and (2) **Pareto** [9]: $doc_score > 1 - np.random.pareto(\alpha)$, $\alpha = 9$. The keeping ratios of our re-implemented GPT-3 quality classifiers are generally in line with the original one, and our Chinese extended version maintains a keeping ratio comparable to that of the English version.

7.2.4 System Scalability. To verify the enhanced scalability of our system (as detailed in Sec. 6), we carry out a series of experiments to measure data processing times across multiple servers. Specifically, we adopt the StackExchange and arXiv datasets from RedPajama. The total size of the StackExchange and arXiv datasets are 65GB and 140GB in jsonl format, respectively. We compare the performance of Data-Juicer on Ray, Data-Juicer on Beam (using the Flink backend), and original Data-Juicer in these tests. More details about the implementation and experimental platforms are in Appendix B.3.5.

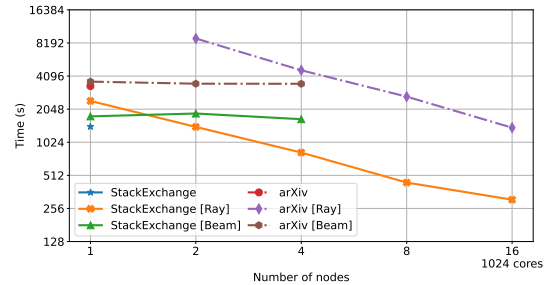


Figure 10: Processing time with varying number of nodes. Data-Juicer accelerates processing in distributed mode.

The experiment results are illustrated in Figure 10. Notably, thanks to various optimizations, our original system outperforms both Ray and Beam in the single server scenario. Moreover, as the number of nodes increases, the processing time of our system on Ray decreases proportionally (up to 87.4% and 84.6% time reduction on StackExchange and arXiv respectively), demonstrating its effective scalability across multiple servers.

Nonetheless, the processing time of Data-Juicer on Beam remains almost unchanged as the number of nodes increases. Upon further investigation of the processing workflow, we found that the limited scalability of Data-Juicer on Beam is primarily constrained by the data loading component of Beam, which leads to a dominant file loading time ratio and requires substantial development changes for adaptation and further performance optimization.

7.3 Empowering Real-world Products

Data-Juicer has been adopted by several real-world LLM-based products, playing a crucial role in data understanding and processing. It evolves continually through the integration of feedback from real-world demands. A notable testament to its utility is its contribution to the development of several industrial LLMs from Alibaba Cloud’s Tongyi suite [21], such as Dianjin, which is used for financial analysis; Zhiwen, a reading assistance tool; and Xingchen, which specializes in AI character customization. Moreover, the data processing capabilities of Data-Juicer have been incorporated into Alibaba Cloud’s Platform for AI (PAI) [22] to support more real-world applications.

Our system’s fine-grained OP abstraction, coupled with the extensive tools for LLM data-processing, empowers users to easily explore and refine data recipes tailored to the distinct textual attributes of diverse use cases. For example, within the financial sector, it is crucial to accommodate data that includes numerous digits and standardized terminology. In the realm of reading assistance, the focus shifts to data characterized by extended text lengths and coherent structures. Conversely, character customization demands data rich in dialogue and varied enough to support personalized services. Data-Juicer adeptly meets these varied demands by facilitating the combination of distinct OPs, hyper-parameters, and tools that adapt to the unique need of each real-world application.

8 CONCLUSIONS

To conclude, the introduction of Data-Juicer reflects a new step forward in the field of data-centric LLM development. By offering a user-friendly, versatile, and efficient solution, Data-Juicer effectively addresses the existing limitations of open-source tools for LLM data processing, which lean towards data reproducibility at the expense of adaptability and usability. The decoupling of traditionally linked components fosters greater abstraction and modularity, and the organic arrangement of over 50 built-in operators, dedicated tools, and abundant data recipes serves diverse needs for LLM pre-training and fine-tuning. Beyond supporting auto-evaluation, Data-Juicer is carefully optimized and seamlessly integrated with both ecosystems for LLM training and evaluation, as well as distributed computing. Empirical validation bears witness to substantial improvements in LLMs’ performance using Data-Juicer’s data recipes, and shows advances in system efficiency and scalability. As such, Data-Juicer stands as a compelling addition to the toolkit for LLM data processing, which we hope can shed light on broader research for the field of data-centric LLM development.

REFERENCES

- [1] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Capelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Lamerat, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. Falcon-40B: an open large language model with state-of-the-art performance. (2023).
- [2] Apache Arrow. 2023. <https://arrow.apache.org/>
- [3] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Benjamin Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. 2021. A General Language Assistant as a Laboratory for Alignment. *CoRR* abs/2112.00861 (2021).
- [4] Stephen H. Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesh Sharma, Taewoon Kim, M. Saiful Bari, Thibault Févry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged Saeed AlShaibani, Shanya Sharma, Urnish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir R. Radev, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. PromptSource: An Integrated Development Environment and Repository for Natural Language Prompts. In *ACL (demo)*, 93–104.
- [5] Apache Beam. 2023. <https://beam.apache.org/>
- [6] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *ICML*, Vol. 202. 2397–2430.
- [7] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An Open-Source Autoregressive Language Model. *CoRR* abs/2204.06745 (2022).
- [8] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-Wise Independent Permutations. *J. Comput. System Sci.* 60, 3 (2000), 630–659.
- [9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.
- [10] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *CoRR* abs/2303.12712 (2023).
- [11] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large Language Models as Tool Makers. *CoRR* abs/2305.17126 (2023).
- [12] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Kostas and batch processing in a single engine. *IEEE Data Eng. Bull.* 38, 4 (2015).
- [13] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramèr, and Chiyuan Zhang. 2023. Quantifying Memorization Across Neural Language Models. In *ICLR*.
- [14] Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *STOC*. 380–388.
- [15] ChatGLM2-6B . 2023. <https://github.com/THUDM/ChatGLM2-6B>
- [16] ChatLLaMA. 2023. <https://github.com/nebulu-ai/nebulu/tree/main/optimization/chatllama>
- [17] Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2023. AlphaGasus: Training A Better Alpaca with Fewer Data. *CoRR* abs/2307.08701 (2023).
- [18] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021).
- [19] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. <https://vicuna.lmsys.org>
- [20] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. 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. *CoRR* abs/2210.11416 (2022).
- [21] Alibaba Cloud. 2023. <https://tongyi.aliyun.com>
- [22] Alibaba Cloud. 2023. <https://www.alibabacloud.com/en/product/machine-learning>
- [23] Yann Collet and Murray Kucherawy. 2021. Zstandard Compression and the 'application/zstd' Media Type. RFC 8878.
- [24] Together Computer. 2023. *RedPajama: An Open Source Recipe to Reproduce LLaMA training dataset*. <https://github.com/togethercomputer/RedPajama-Data>
- [25] Michael J Cormier, Jonathan R Belyeu, Brent S Pedersen, Joseph Brown, Johannes Köster, and Aaron R Quinlan. 2021. Go Get Data (GGD) is a framework that facilitates reproducible access to genomic data. *Nature Communications* 12, 1 (2021), 2151.
- [26] Common Crawl. 2023. <https://commoncrawl.org/>
- [27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*, 4171–4186.
- [28] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen S. Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts. In *ICML*. 5547–5569.
- [29] EleutherAI. 2023. Pythia-1.4B. <https://huggingface.co/EleutherAI/pythia-1.4b>
- [30] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards Revealing the Mystery behind Chain of Thought: a Theoretical Perspective. *CoRR* abs/2305.15408 (2023).
- [31] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2021. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *CoRR* abs/2101.00027 (2021).
- [32] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. *A framework for few-shot language model evaluation*.
- [33] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models. In *EMNLP (Findings)*, 3356–3369.
- [34] Xinyang Geng and Hao Liu. 2023. *OpenLLaMA: An Open Reproduction of LLaMA*. https://github.com/openlm-research/open_llama
- [35] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need. arXiv:2306.11644 [cs.CL]
- [36] Project Gutenberg. 2023. <https://www.gutenberg.org/>
- [37] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.
- [38] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings. *CoRR* abs/2305.11554 (2023).
- [39] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *ICLR*.
- [40] Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural Instructions: Tuning Language Models with (Almost) No Human Labor. *CoRR* abs/2212.09689 (2022).
- [41] Technology Innovation Institute. 2023. Falcon-RW-1B. <https://huggingface.co/tiiuae/falcon-rw-1b>
- [42] Technology Innovation Institute. 2023. Falcon-RW-1B. <https://huggingface.co/FlagAlpha/Atom-7B>

- [43] Gautier Izacard, Patrick S. H. Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Few-shot Learning with Retrieval Augmented Language Models. *CoRR abs/2208.03299* (2022).
- [44] Abhinav Jain, Hima Patel, Lokesh Nagalapati, Nitin Gupta, Sameep Mehta, Shannukha Guttula, Shashank Mujumdar, Shazia Afzal, Ruhi Sharma Mittal, and Vitobha Munigala. 2020. Overview and importance of data quality for machine learning tasks. In *KDD*. 3561–3562.
- [45] Yunjie Ji, Yong Deng, Yan Gong, Yiping Peng, Qiang Niu, Baochang Ma, and Xiangang Li. 2023. BELLE: Be Everyone's Large Language model Engine. <https://github.com/LianjiaTech/BELLE>.
- [46] jsonargparse. 2023. <https://github.com/omni-us/jsonargparse>
- [47] Nikhil Kandpal, Eric Wallace, and Colin Raffel. 2022. Deduplicating Training Data Mitigates Privacy Risks in Language Models. In *ICML*. 10697–10707.
- [48] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnab Danturli, Andrew Maguire, Christoph Schuhmann, Hru Nguyen, and Alexander Mattick. 2023. OpenAssistant Conversations - Democratizing Large Language Model Alignment. *CoRR abs/2304.07327* (2023).
- [49] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP*.
- [50] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP (Demonstration)*.
- [51] Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Vilanova del Moral, Teven Le Scao, Leandro von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, Jörg Froberg, Mario Sasko, Quentin Lhoest, Angelina McMillan-Major, Gérard Dupont, Stella Biderman, Anna Rogers, Loubna Ben Allal, Francesco De Toni, Giada Pistilli, Olivier Nguyen, So-maieh Nikpoor, Maraim Masoud, Pierre Colombo, Javier de la Rosa, Paulo Villegas, Tristan Thrush, Shayne Longpre, Sebastian Nagel, Leon Weber, Manuel Muñoz, Jian Zhu, Daniel van Strien, Zaid Alyafeai, Khalid Almubarak, Minh Chien Vu, Itziar González-Dios, Aitor Soroa, Kyle Lo, Manan Dey, Pedro Ortiz Suarez, Aaron Gokaslan, Shamik Bose, David Ifeoluwa Adelan, Long Phan, Hieu Tran, Ian Yu, Suhas Pai, Jenny Chim, Violette Lepercq, Suzana Ilic, Margaret Mitchell, Alexandra Sasha Luccioni, and Yacine Jernite. 2022. The BigScience ROOTS Corpus: A 1.6TB Composite Multilingual Dataset. In *NeurIPS*.
- [52] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating Training Data Makes Language Models Better. In *ACL (1)*. 8424–8445.
- [53] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *EMNLP (1)*. 3045–3059.
- [54] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL*. 7871–7880.
- [55] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Sasko, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander M. Rush, and Thomas Wolf. 2021. Datasets: A Community Library for Natural Language Processing. In *EMNLP (Demos)*. 175–184.
- [56] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* 18 (2017), 185:1–185:52.
- [57] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *CoRR abs/2305.06161* (2023).
- [58] Yunxiang Li, Zihan Li, Kai Zhang, Ruilong Dan, and You Zhang. 2023. ChatDoc-tor: A Medical Chat Model Fine-tuned on LLaMA Model using Medical Domain Knowledge. *CoRR abs/2303.14070* (2023).
- [59] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shiban Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic Evaluation of Language Models. *CoRR abs/2211.09110* (2022).
- [60] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. *CoRR abs/2303.16634* (2023).
- [61] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The Flan Collection: Designing Data and Methods for Effective Instruction Tuning. *CoRR abs/2301.13688* (2023).
- [62] Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, and Daphne Ippolito. 2023. A Pretrainer's Guide to Training Data: Measuring the Effects of Data Age, Domain Coverage, Quality, & Toxicity. *CoRR abs/2305.13169* (2023).
- [63] Ilya Loshchilov and Frank Hutter. 2017. Fixing Weight Decay Regularization in Adam. *CoRR abs/1711.05101* (2017).
- [64] LZ4. 2023. <https://www.lz4.org/>
- [65] Kamil Malinka, Martin Peresini, Anton Firc, Ondrej Hujnak, and Filip Janus. 2023. On the Educational Impact of ChatGPT: Is Artificial Intelligence Ready to Obtain a University Degree? *CoRR abs/2303.11146* (2023).
- [66] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *OSDI*. 561–577.
- [67] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *ICLR*.
- [68] OpenAI. 2022. Our approach to alignment research. *OpenAI Blog* (August 2022).
- [69] OpenAI. 2023. GPT-4 Technical Report. *CoRR abs/2303.08774* (2023).
- [70] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.
- [71] 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. *CoRR abs/2306.01116* (2023).
- [72] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *NAACL-HLT*. 2227–2237.
- [73] Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. Reasoning with Language Model Prompting: A Survey. [arXiv:2212.09597](https://arxiv.org/abs/2212.09597) [cs.CL]
- [74] Zheng Lin Qingyi Si. 2023. Alpaca-CoT: An Instruction Fine-Tuning Platform with Instruction Data Collection and Unified Large Language Models Interface. <https://github.com/PhoebusSi/alpaca-CoT>
- [75] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).
- [76] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [77] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* (2020), 140:1–140:67.
- [78] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD*. 3505–3506.
- [79] Xiaozhe Ren, Pingyi Zhou, Xinfan Meng, Xinjing Huang, Yadao Wang, Weichao Wang, Pengfei Li, Xiaoda Zhang, Alexander Podolskiy, Grigory Arshinov, Andrey Bout, Irina Piontkovskaya, Jiانشeng Wei, Xin Jiang, Teng Su, Qun Liu, and Jun Yao. 2023. PanGu- Σ : Towards Trillion Parameter Language Model with

- Sparse Heterogeneous Computing. *CoRR* abs/2303.10845 (2023).
- [80] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Galle, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muenninghoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Launay, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *CoRR* abs/2211.05100 (2022).
- [81] Omid Shahmirzadi, Adam Lugowski, and Kenneth Young. 2019. Text similarity in vector space models: a comparative study. In *ICMLA*. 659–666.
- [82] Bobak Shahrhiri, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
- [83] Noam Shazeer. 2020. GLU Variants Improve Transformer. abs/2002.05202 (2020).
- [84] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580* (2023).
- [85] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019).
- [86] Soldaini, Luca and Lo, Kyle and Kinney, Rodney and Naik, Aakanksha and Ravichander, Abhilasha and Bhagia, Akshita and Groeneveld, Dirk and Schwenk, Dustin and Magnusson, Ian and Chandu, Khyathi. 2023. *The Dolma Toolkit*. Apache 2.0 License, Version 0.9.0, <https://github.com/allenai/dolma>.
- [87] Streamlit. 2023. <https://streamlit.io/>
- [88] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. RoFormer: Enhanced Transformer with Rotary Position Embedding. *CoRR* abs/2104.09864 (2021).
- [89] Yixuan Su, Tian Lan, Huayang Li, Jialu Xu, Yan Wang, and Deng Cai. 2023. PandaGPT: One Model To Instruction-Follow Them All. *CoRR* abs/2305.16355 (2023).
- [90] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiayang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, Weixin Liu, Zhihua Wu, Weibao Gong, Jianzhong Liang, Zhizhou Shang, Peng Sun, Wei Liu, Xuan Ouyang, Dianhai Yu, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation. *CoRR* abs/2107.02137 (2021).
- [91] Zhongxiang Sun. 2023. A Short Survey of Viewing Large Language Models in Legal Aspect. *CoRR* abs/2303.09136 (2023).
- [92] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.
- [93] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Théo Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023).
- [94] Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. 2022. What Language Model Architecture and Pretraining Objective Works Best for Zero-Shot Generalization?. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*. 22964–22984.
- [95] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, and Xudong Shen. 2022. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks. In *EMNLP*. 5085–5109.
- [96] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned Language Models are Zero-Shot Learners. In *ICLR*.
- [97] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *CoRR* abs/2206.07682 (2022).
- [98] Jerry W. Wei, Le Hou, Andrew K. Lampinen, Xiangning Chen, Da Huang, Yi Tay, Xinyun Chen, Yifeng Lu, Denny Zhou, Tengyu Ma, and Quoc V. Le. 2023. Symbol tuning improves in-context learning in language models. *CoRR* abs/2305.08298 (2023).
- [99] Xiang Wei, Xingyu Cui, Ning Cheng, Xiaobin Wang, Xin Zhang, Shen Huang, Pengjun Xie, Jinan Xu, Yufeng Chen, Meishan Zhang, Yong Jiang, and Wenjuan Han. 2023. Zero-Shot Information Extraction via Chatting with ChatGPT. *CoRR* abs/2302.10205 (2023).
- [100] Wikipedia. 2023. https://en.wikipedia.org/wiki/Main_Page
- [101] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP (Demos)*. 38–45.
- [102] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David S. Rosenberg, and Gideon Mann. 2023. BloombergGPT: A Large Language Model for Finance. *CoRR* abs/2303.17564 (2023).
- [103] Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. 2023. DoReMi: Optimizing Data Mixtures Speeds Up Language Model Pretraining. *CoRR* abs/2305.10429 (2023).
- [104] Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. 2023. FinGPT: Open-Source Financial Large Language Models. *CoRR* abs/2306.06031 (2023).
- [105] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufeı Xue, Jidong Zhai, Wenguang Chen, Peng Zhang, Yuxiao Dong, and Jie Tang. 2022. GLM-130B: An Open Bilingual Pre-trained Model. abs/2210.02414 (2022).
- [106] Biao Zhang and Rico Sennrich. 2019. Root Mean Square Layer Normalization. In *NeurIPS*. 12360–12371.
- [107] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *CoRR* abs/2205.01068 (2022).
- [108] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. *CoRR* abs/2303.18223 (2023).
- [109] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. *CoRR* abs/2304.06364 (2023).
- [110] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In *ICCV*. 19–27.

APPENDIX OF DATA-JUICER: A ONE-STOP DATA PROCESSING SYSTEM FOR LARGE LANGUAGE MODELS

A ADDITIONAL DETAILS OF DATA-JUICER

A.1 Base Classes of OPs in Data-Juicer

We illustrate the core base classes of operators (OPs) in Data-Juicer at listing 1.

A.2 Theoretical Analysis of Space Usage for Caches and Checkpoints

Caches are generated after some of the functions of Dataset, such as map, filter. Generally, caches can be categorized into cache data and indices. The total size of a set of indices is very small so we can ignore these parts when conducting the space usage analysis. On the contrary, the size of the cache data is nearly the same as the input dataset. Here we assume that the sizes of cache data and checkpoints are all the same as the input dataset’s size. And there must be one cache data file for the original dataset after it’s loaded.

Assume that there are M Mappers, F Filters, and D Deduplicators in the processing configuration, and the size of the original dataset is S , the detailed analysis for cache mode and checkpoint mode is shown below.

Space Usage of Cache Mode. Caches are generated after each OP. Mappers, Filters, and Deduplicators only generate one set of cache data. Besides, the first Filter would generate an extra set of cache data because a new column for storing statistics will be added to the dataset. Therefore the total disk space usage of caches is:

$$Space_{[cache_mode]} = (1 + M + F + \mathbb{I}(F > 0) + D) \times S,$$

where $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 when \cdot is true, otherwise returns 0.

Space Usage of Checkpoint Mode. Checkpoints are only generated when any exception or error occurs. However, caches are still stored after disabling the cache mode due to the features of Dataset. We clean up older caches after each OP. The detailed cleanup pipeline is: 1). OP_i finished, 2). caches for OP_i generated, 3). caches for OP_{i-1} cleaned up. Thus there exists at most two sets of caches at the same time theoretically in step 2. Considering the caches of the original dataset, the peak disk space usage of caches in checkpoint mode is:

$$Space_{[checkpoint_mode]} = 3 \times S.$$

B ADDITIONAL NUMERICAL RESULTS

Table 5: Evaluation results of three types of quality classifiers.

| Quality Classifier | Precision | Recall | F1 |
|--------------------|-----------|--------|--------|
| GPT-3 | 96.82% | 98.14% | 97.47% |
| CHINESE | 98.00% | 99.30% | 98.64% |
| CODE | 71.23% | 54.21% | 61.56% |

```
1 class Formatter:
2     ...
3     def load_dataset(self, *args) -> Dataset:
4         ...
5     ...
6
7 class Mapper:
8     ...
9     def process(self, sample: Dict) -> Dict:
10        ...
11    ...
12
13 class Filter:
14    ...
15    def compute_stats(self, sample: Dict) -> Dict:
16        ...
17
18    def process(self, sample: Dict) -> bool:
19        ...
20    ...
21
22 class Deduplicator:
23    ...
24    def compute_hash(self, sample: Dict) -> Dict:
25        ...
26
27    def process(self, dataset: Dataset) -> Dataset:
28        ...
29    ...
```

Listing 1: The illustration of OP base classes in Data-Juicer.

B.1 Quality Classifier

Firstly, we will show how we can reproduce the GPT-3 and achieve comparable performance.

We follow the training procedure of quality classifier in GPT-3 [9] that used a logistic regression classifier with features from standard tokenizer and HashingTF of PySpark. Based on this, we expand this training pipeline to Chinese text and various code types. The training details are listed in Table 6, where the keeping method includes:

- label: $doc_score > 0.5$
- pareto [9]: $doc_score > 1 - np.random.pareto(\alpha), \alpha = 9$

We split these datasets into training and evaluation splits with a split ratio of 4:1. Then these classifiers trained on the training split are evaluated on the evaluation split. Experimental results are shown in Table 5. As we can see, reproduced GPT-3 and its Chinese version perform well except for the Code version. We speculate that the positive and negative splitting method for Code quality classifier now might not be a good choice, and we leave this issue to future research.

Besides, we compare keeping ratios when using these classifiers to re-sample CommonCrawl between the original GPT-3 quality classifier and our reproduced classifiers, which is shown in Table 4. The keeping ratio of the original GPT-3 quality classifier is estimated by the data size before and after filtering described in GPT-3 paper [9]. We can see that the keeping ratios of our reproduced GPT-3 quality classifiers are basically aligned with the original one.

B.2 Data Recipes

For pre-training data, we acquired a vast amount of raw textual corpora primarily following the procedural guidelines of RedPajama [24] and the Pile [31]. The common subsets were merged and

Table 6: Training configuration of 3 types of quality classifiers.

| Quality Classifier | Tokenizer | Keep Method | Positive Datasets | Negative Datasets |
|--------------------|--------------------|-------------|--|--|
| GPT-3 | Standard Tokenizer | pareto | Wikipedia-en & books1 & OpenWebText2 | CommonCrawl |
| CHINESE | Sentencepiece | label | Wikepeida-zh & Wudao | Samples in Chinese from CommonCrawl |
| CODE | Sentencepiece | label | Samples with max_stars_count>=1372 from TheStack | Random Samples from the rest of TheStack |

subjected to Data-Juicer refinements. The resultant data recipe is presented in Table 7, which covers 15 prominent components. We use the SentencePiece [50] tokenizer as implemented in GPT-NeoX-20B [7] to prepare text and report the counted number of tokens. The sampling proportion is the normalization of token numbers, except for Books and Wikipedia, which undergo 2 and 2.5 epochs respectively, to enhance the weighting of high-quality corpora.

Table 7: Statistics of Data-Juicer’s pre-training data.

| Component | #Tokens | Sampling prop. |
|------------------|-----------------|----------------|
| CommonCrawl | 360,925,581,674 | 44.91% |
| C4 | 181,951,688,729 | 22.64% |
| GitHub | 65,076,921,292 | 8.10% |
| Books | 26,389,944,579 | 6.57% |
| Wikipedia | 17,615,935,449 | 5.48% |
| arXiv | 29,093,082,586 | 3.62% |
| PubMed Central | 25,589,708,647 | 3.18% |
| StackExchange | 19,793,629,900 | 2.46% |
| FreeLaw | 13,057,506,102 | 1.62% |
| PubMed Abstracts | 5,208,343,613 | 0.65% |
| USPTO | 4,021,281,155 | 0.50% |
| EuroParl | 780,962,770 | 0.10% |
| HackerNews | 485,584,871 | 0.06% |
| PhilPapers | 478,040,431 | 0.06% |
| HIH ExPorter | 436,414,852 | 0.05% |

For fine-tuning data, we merge and refine tens of Alpaca-CoT datasets. Each dataset can be categorized into English, Chinese and Multilingual by language; into instruct fine-tuning, and chat fine-tuning including single-round dialog, multi-round dialog and preference by usage; multi-task and task-specific by task type; and human-generated, self-instruct, and mixed collection of datasets by the generation method. The detailed numbers of datasets for each category are presented in Table 8.

More information about these datasets can be found on the Data-Juicer recipes page² of our repository.

²https://github.com/alibaba/data-juicer/blob/main/configs/data_juicer_recipes

Table 8: Statistics of Data-Juicer fine-tuning data used in our experiments. *These tags are newly added by Data-Juicer compared to the original tag sets of Alpaca-CoT [74]. “CFT” indicates Chat Fine-Tuning.

| Category | Sub-Category | #Datasets |
|-------------------|----------------------------|-----------|
| Language | English | 28 |
| | Chinese | 14 |
| | Multilingual | 3 |
| Usage* | Instruct Fine-Tuning (IFT) | 17 |
| | CFT: Single-Round Dialog | 23 |
| | CFT: Multi-Round Dialog | 2 |
| Task Type | CFT: Preference | 5 |
| | Multi-Task | 27 |
| Generation Method | Task-Specific | 13 |
| | Human-Generated | 3 |
| | Self-Instruct | 12 |
| | Mixed | 5 |
| | Collection of Datasets | 19 |

B.3 Experiments Details

B.3.1 Models and Training For Pre-training Data. We adhere to the official paper [93] and leverage open-source implementation [34] to build standard LLaMA models. Basically, it is to apply RMSNorm [106], the SwiGLU activation [83], and rotary positional embedding [88] on the decoder-only transformer architecture. The LLaMA-1.3B model is composed of 24 transformer layers, each with 16 self-attention heads and 2048 bottleneck units.

LLMs are pre-trained using the AdamW optimizer [63] with hyper-parameters $\beta_1 = 0.9$ and $\beta_2 = 0.95$. For LLaMA-1.3B, the initial learning rate gradually increases to $2e-5$ using 1% warm-up steps and finally decays to 10% through a cosine schedule. The weight decay is set to 0.1 and the gradient ℓ_2 -norm is clipped to 1.0.

B.3.2 Models and Training of Fine-Tuning Data. In fine-tuning, we choose LLaMA-7B as our basic model and fine-tuned it for 3 epochs. We follow the hyper-parameter settings in Alpaca [92]. Specifically, the optimizer is AdamW with a learning rate of $2e-5$,

global batch size of 256, and weight decay of 0. The learning rate schedules in a cosine style with 3% initial warm-up steps.

Regarding the data recipes in Table 3, for (CFT, EN) case, we consider 5 competitive subsets (Alpaca, GPTeacher, FastChat, Guanaco, and CodeAlpaca) from Alpaca-CoT as candidate datasets; for (CFT, ZH) case, we use (AlpacaGPT4, Belle, Instinwild) as candidate datasets. Generally speaking, we bucket from these candidate datasets according to more than a dozen built-in analytical dimensions, sampling a fixed amount of data from each dimension to increase the diversity of the processed data as appropriately as possible. More detailed hyper-parameters of data processing can be found in our released data recipes.

Both the pre-trained and fine-tuned reference models are released in our homepage.

B.3.3 System Performance Experiments. The experiments of end-to-end processing mentioned in section 7.2.1 are all conducted on the same machine with 128 cores of Intel(R) Xeon(R) Platinum 8369B models and about 990GB memory. Before starting these experiments, the original datasets, third-party models, and other assets will be prepared in advance for both baselines and Data-Juicer, and the intermediate cache files will be cleaned after every complete process for Data-Juicer. After processing, we use the same number of processes for processing the dataset to export the result dataset to the local SSD.

As for the resource monitoring tool, it’s implemented based on the PSUTIL³ library. It samples the memory for all related processes every second during the processing pipeline. Then we compute the average memory usage by summing the memory usage over all processes and dividing by the number of processes used in each experiment. Finally, we aggregate all data and compute the average memory usage over time.

B.3.4 End-to-end System Baselines. We mainly compared the end-to-end system performance between our Data-Juicer and two state-of-the-art baselines in the above experiments w.r.t system performance: RedPajama [24] and Dolma [86]. Besides the empirical comparison in Sec.7.2.1, here we make more detailed introduction and comparison about them.

RedPajama.⁴ The RedPajama project, developed by Together AI, initially aims to reproduce the LLaMA training dataset [93] and open-source the entire code for data collection and processing, making it a significant and popular contribution to the LLM community. This is the primary reason for selecting it as our baseline. RedPajama provides a reproduced version of all seven subsets of the LLaMA training dataset, including arXiv, Books, C4, Common-Crawl, GitHub Code, Stack Exchange, and Wikipedia.

While RedPajama has made valuable contributions, our work explores different aspects and offers complementary features. For instance: (1) RedPajama’s design is closely tied to specific datasets, which present challenges for adapting its data processing pipelines to other datasets. (2) Its focus on reproducing the LLaMA datasets lead to trade-offs in efficiency, which is not the primary concern of the RedPajama project. (3) The current data processing component in RedPajama lacks systematization and customization. Adding new

data processing methods to the existing pipelines would require understanding and modifying a significant portion of the code. As a result, most users typically opt to utilize the RedPajama Dataset directly rather than attempting to customize or improve its data processing pipelines.

Dolma.⁵ The Dolma project, originating from Allen AI, comprises two components: the Dolma Dataset and the Dolma Toolkit. It is also a newly established data processing initiative. We selected the Dolma Toolkit as a baseline because its objective of generating pre-training data for language modeling aligns with one of our target data types (we focus on both pre-training and fine-tuning data). The toolkit offers numerous “Taggers” that enable attribute tagging (analogous to ‘stats’ in Data-Juicer) for each document sample. These tags are then used to filter out samples with undesirable attributes. Users have the flexibility to create custom taggers tailored to their specific needs.

However, we encountered several limitations when using Dolma for dataset processing. Firstly, Dolma’s workflow involves multiple stages—tagging, deduplication, mixing, and various configurations—lacking support for an end-to-end data processing pipeline. Secondly, to leverage high-performance parallel processing, users are required to partition the input dataset into multiple shards in advance, incurring additional overhead. Thirdly, Dolma imposes certain requirements on input datasets, such as mandatory fields and a specific directory structure, necessitating further preprocessing before use. Moreover, it restricts input formats to JSONL or its gzipped variant. These constraints diminish the toolkit’s flexibility, thereby increasing the cost of use and rendering the Dolma Toolkit relatively less user-friendly.

B.3.5 Scalability. Our experiments are performed on a platform comprising 16 servers, each equipped with a 64-core Intel(R) Xeon(R) Platinum CPU (mix of 8269CY and 8163 models) and 512 GB of memory. The network bandwidth shared among these servers is 20 Gbps. We utilize NAS storage to house both the raw data and the processed results. For the scalability experiments, we consider the two baselines as follows:

- **Data-Juicer on Ray:** We implement a Ray [66] executor for Data-Juicer, which only adapts the underlying interfaces of the HuggingFace-datasets with Ray-datasets, while all OPs of Data-Juicer remain unchanged. This implies that users’ code based on our native Python version can be seamlessly migrated from a single-machine version to distributed computing environments.

- **Data-Juicer on Beam:** This method is based on Apache Beam with the Apache Flink Runner. When compared to the Ray version, the Beam version requires additional code development to meet the demands of the Beam data processing pipeline. This includes the adaptations of several OPs and the replacement of the Formatter/Exporter with counterparts in Beam.

B.4 Per-Task Evaluation

For a thorough and consolidated assessment, we have summarized the individual scores of evaluated LLMs on the 16 core HELM assessment tasks in Table 9.

³<https://github.com/giampaolo/psutil>

⁴We compared RedPajama in our experiments with its github commit ID as: 45b37c2a1d1e495b0f48549ef3ce03ff029f7881.

⁵We compared Dolma in our experiments with its github commit ID as: 5a010a2685914b1db7744426abfb4b9ece52da95.

Table 9: Evaluation results on 16 core tasks of HELM benchmark.

| Task | Falcon-1.3B | Pythia-1.4B | LLaMA-1.3B (Data-Juicer) | LLaMA-1.3B (Data-Juicer IFT) |
|--------------------------------|--------------------|--------------------|-------------------------------------|---|
| MMLU | 24.7 | 26.0 | 25.9 | 27.0 |
| BoolQ | 63.0 | 56.0 | 49.0 | 56.0 |
| NarrativeQA | 32.1 | 31.5 | 38.2 | 49.9 |
| NaturalQuestions (closed-book) | 10.7 | 10.5 | 10.1 | 11.2 |
| NaturalQuestions (open-book) | 50.0 | 49.8 | 45.9 | 54.3 |
| QuAC | 24.3 | 26.5 | 26.0 | 21.7 |
| HellaSwag | 67.0 | 57.0 | 56.0 | 52.0 |
| OpenbookQA | 44.0 | 34.0 | 40.0 | 43.0 |
| TruthfulQA | 19.0 | 21.0 | 33.0 | 33.0 |
| MS MARCO (regular) | 16.8 | 12.9 | 11.2 | 12.1 |
| MS MARCO (TREC) | 33.5 | 27.4 | 26.9 | 28.1 |
| IMDB | 55.0 | 84.0 | 80.0 | 84.0 |
| XSUM | 5.7 | 6.5 | 5.2 | 5.3 |
| CNN/DailyMail | 4.0 | 8.4 | 7.8 | 11.1 |
| CivilComments | 49.4 | 49.7 | 50.1 | 50.0 |
| RAFT | 44.3 | 42.3 | 42.1 | 49.3 |