A Survey of LLM × DATA

Xuanhe Zhou*¶, Junxuan He*¶, Wei Zhou*¶, Haodong Chen*¶, Zirui Tang*¶, Haoyu Zhao*¶, Xin Tong*, Guoliang Li†, Youmin Chen*, Jun Zhou*, Zhaojun Sun*, Binyuan Hui‡, Shuo Wang†, Conghui He§, Zhiyuan Liu†, Jingren Zhou‡, Fan Wu*
*Shanghai Jiao Tong University †Tsinghua University ‡Alibaba Group §Shanghai AI Laboratory
https://github.com/weAIDB/awesome-data-llm

Abstract—The integration of large language model (LLM) and data management (DATA) is rapidly redefining both domains. In this survey, we comprehensively review the bidirectional relationships. On the one hand, **DATA4LLM**, spanning large-scale data processing, storage, and serving, feeds LLMs with diversity, redundant, high quality, and sanitized data (following the "**IaaS**" concept) required for stages like pre-training, post-training, retrieval-augmented generation, and agentic workflows: (i) Data processing for LLMs includes scalable acquisition, deduplication, filtering, selection, domain mixing, and synthetic augmentation; (ii) Data Storage for LLMs focuses on efficient data and model formats, distributed and heterogeneous storage hierarchies, KV-cache management, and fault-tolerant checkpointing; (iii) Data serving for LLMs tackles challenges in RAG (e.g., knowledge post-processing), LLM inference (e.g., prompt compression, data provenance), and training strategies (e.g., data packing and shuffling). On the other hand, in **LLM4DATA**, LLMs are emerging as general-purpose engines for data management. We review recent advances in (i) data manipulation, including automatic data cleaning, integration, discovery; (ii) data analysis, covering reasoning over structured, semi-structured, and unstructured data, and (iii) system optimization (e.g., configuration tuning, query rewriting, anomaly diagnosis), powered by LLM techniques like retrieval-augmented prompting, task-specialized fine-tuning, and multi-agent collaboration.

Index Terms—Large Language Model, Data Management, DATA4LLM, LLM4DATA

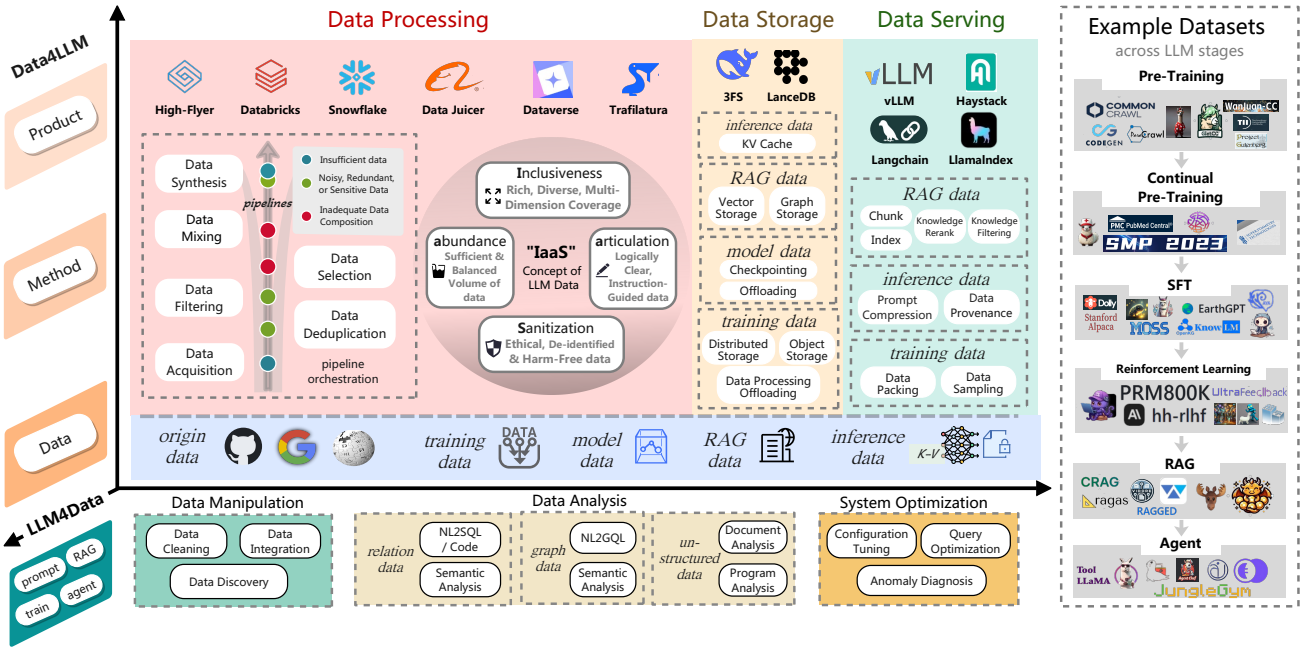

Fig. 1: Overview of LLM × DATA (with "**IaaS**" Concept).

1 INTRODUCTION

LARGE language models (LLMs[1]) have made remarkable progress in both *general domain applications* (e.g., open-domain question answering [332], cross-modal video summarization [175], general-purpose code generation [191]) and *specific domain applications* (e.g., biomedical literature analysis [394], legal document review [221], SQL generation for business intelligence [250]). As shown in Figure 1, apart from technical advances in LLMs [289], [64], [460], [301], [241], [227], data management has emerged as a critical factor in unlocking LLMs' full potential in these applications (DATA4LLM). It includes efficient and scalable solutions for data processing, storage, and serving across the LLM lifecycle, as evidenced in recent academic studies [157], [285], [254] and industry reports [327], [433], [69], [39]. Conversely,

¶ *Co-first authors with equal contributions.*

1. We use LLMs to refer to billion-scale language models capable of supporting general NLP tasks [472] or multimodal tasks [444], [322].

LLM-powered techniques are increasingly being adopted to enhance data management tasks, such as data manipulation, analysis, and system optimization (LLM4DATA).

DATA4LLM. Effective data management is fundamental to the scalable development and deployment of LLMs. To illustrate this, we highlight representative scenarios where LLMs depend on specialized techniques for data processing, storage, and serving across various stages of the LLM lifecycle.

Example-① Data Processing for LLMs. Processing a large-scale training dataset (e.g., ∼4 TB multi-modal tokens utilized in Qwen2.5-VL pretraining [70]) poses several challenges. First, acquiring diverse raw data (e.g., over 10,000 object categories for visual grounding) demands substantial efforts in data collection (Section 2.3.1) and, in many cases, data synthesis (Section 2.3.6). Second, preparing high-quality training samples requires robust pre-processing, including rigorous data filtering (Section 2.3.3), along with dedicated evaluation approaches. Third, the overall performance of LLMs depends heavily on an end-to-end pipeline that effectively schedules and coordinates these processing tasks, especially for the pretraining stage (Section 2.3.7).

Example-② Data Storage for LLMs. Managing storage for LLMs, spanning both training datasets (see Example-①) and massive model parameters (e.g., DeepSeek-R1 with 671B parameters [162]), poses significant challenges. First, large-scale datasets must be partitioned and distributed across multiple storage nodes, introducing challenges in data placement and consistency management (Section 2.4.2). Second, to support efficient LLM training and inference, these storage nodes must deliver high I/O throughput for timely data transfer to compute nodes (Section 2.4.4). Third, the massive size of model parameters increases the risk of training interruptions, necessitating robust fault tolerance mechanisms to recover and resume training from intermediate states (Section 2.4.5).

Example–③ Data Serving for LLMs. Data serving plays a critical role in selecting and preparing input data (e.g., the task-specific prompts), directly affecting the quality of LLM's responses. Taking retrieval-augmented generation (RAG) as an example, EyeLevel.ai [37] observed that when relying solely on vector similarity, RAG accuracy declines notably with 10,000-page documents, and the performance degradation can reach up to 12% with 100,000 pages (still fewer than enterprise-scale datasets). Several challenges arise in this context. First, the retrieved knowledge is typically noisy and must be filtered and re-ranked to ensure relevance and factual accuracy (Section 2.5.1). Second, the retrieved content is often lengthy and exceeds the input capacity or comprehension of LLMs, necessitating effective compression techniques to preserve utility while improving performance (Section 2.5.2).

LLM4DATA. Conversely, various LLM-based techniques can be leveraged to enhance core data management tasks, including data manipulation, data analysis, and system-level optimization. The following examples illustrate how LLMs can be applied to improve these tasks in practice.

Example-① LLM-based Data Manipulation. Data manipulation, including cleaning, integration, and discovery, is critical for ensuring high-quality datasets. Traditional methods depend on rigid rules and domain-specific configurations, requiring extensive manual efforts and struggling with complex data samples [243], [78], [74]. For instance, standardizing date formats (e.g., "Fri Jan 1st 10:36:28 2021" vs. "1996.07.10 AD at 15:08:56") or resolving textual inconsistencies (e.g., "Monticello VA, Jasper" vs. "Monticello VAA") typically requires intricate programming scripts or handcrafted constraints [319], [432]. These approaches also struggle with cross-row error detection, such as mismatched city-state-zip entries. In contrast, LLMs can infer semantic similarities and autonomously generate cleaning workflows to resolve such inconsistencies without requiring explicit rule definitions [237], [432], [454]. This semantic understanding enables LLMs to adapt flexibly to diverse data issues and support more scalable and context-aware data manipulation (Section 3.1).

Example-② LLM-based Data Analysis. Data analysis over heterogeneous sources, such as medical records and transactional data, is essential in many real-world applications. Traditional deep learning models, while effective at performing specific semantic-level analysis, struggle to generalize across diverse data formats and task types. For instance, tasks such as table extraction and table-based question answering across heterogeneous sources (e.g., relational tables and knowledge graphs) often require the development of separate, specialized models. This process is both resource-intensive and difficult to scale. In contrast, LLMs offer a unified reasoning framework that leverages broad semantic understanding, enabling them to support a wide range of analytical tasks across various data modalities with greater flexibility and reduced efforts for task-specific engineering (Section 3.2).

Example-③ LLM-based System Optimization. System optimization entails configuring parameters (e.g., memory settings) and monitoring runtime status (e.g., resource utilization) to ensure optimal system performance. Traditional approaches, such as manual tuning or deep learning-based methods, are time-consuming and inefficient [474]. For instance, methods of Bayesian Optimization (BO) or Reinforcement Learning (RL) require numerous workload replays over 20 hours to identify promising configurations for a single TPC-H workload [177]. Moreover, root cause analysis over anomalies can be error-prone, particularly in multi-cause scenarios where metrics are highly interdependent [490]. In contrast, LLMs offer a new paradigm by integrating domain knowledge (e.g., tuning manuals) and applying advanced reasoning to instruct optimization. By leveraging retrieval-augmented prompts, LLMs can efficiently identify root causes or recommend precise configurations, enabling faster and more accurate optimization in complex environments [489], [248], [223] (Section 3.3).

1.1 Techniques of DATA4LLM

Characteristics of LLM Datasets (§ 2.2). As shown in Figure 1, datasets (following the "**IaaS**" concept) play a critical role in enabling the desired capabilities at each LLM stage, including (1) pre-training, (2) continual pre-training, (3) fine-tuning, (4) reinforcement learning, (5) retrieval-augmented generation (RAG), (6) LLM agents, and (7) evaluation. For each stage, we separately analyze the characters of required data (e.g., preferred formats and emphasized aspects within **IaaS**) and the corresponding data techniques (see Table 1).

Data Processing for LLMs (§ 2.3). We introduce techniques to prepare high-quality datasets for LLMs based on a

series of processing steps.

• *Data Acquisition.* Data acquisition aims to (1) extract relevant data (e.g., text and images) from noisy data sources with certain structures (e.g., dynamically rendered web pages) [73], [144], [76], [73], [6], [19], [30], [31], and (2) extract data from complicated data sources (e.g., scanned or handwritten documents) with techniques such as complex layout analysis [202], [18], [392], [180], [391], [407], [257], [326], [406].

• *Data Deduplication.* Data deduplication aims to identify duplicates in large-scale textual or multi-modal data, including exact string matching [122], [299], hash identification [88], [81], [122], [299], [347], [358], [207], [298], sample reweighing [167] and embedding-based clustering [46], [385], [360].

• *Data Filtering.* We review data filtering methods at two primary levels: (1) Sample-level filtering selects high-quality and diverse samples using strategies like perplexity measuring [383], [61], [288], influence assessment [254], [168], clustering methods [45], [436], prompt-based scoring [411], [264], [345], or mixes of these strategies [285], [84], [126]; (2) Content-level filtering aims to remove undesirable or harmful content from large-scale datasets, such as toxic language, personal identifiable information (PII), biased statements [268], [275], and improper images and videos [437], [216], [390].

• *Data Selection.* Data selection aims to select sub-datasets and evaluate their ability to accurately represent the target distribution, especially when handling diverse datasets or domains. There are methods like similarity-based data selection [423], [421], [321], [80], optimization-based data selection [130], [417], [269], and model-based data selection [465].

• *Data Mixing.* Data mixing aims to effectively integrate datasets from diverse domains without degrading quality or destabilizing LLM performance. Key techniques include: (1) *Heuristic optimization*, which empirically tunes data ratios to enhance downstream performance. Examples include two-stage mixing [139], source rebalancing [347], and entropy-based weighting [152]; (2) *Bilevel optimization*, which formulates data weighting as a nested optimization problem to jointly balance training and validation objectives [302], [135]; (3) *Distributionally robust optimization*, which enhances resilience to worst-case domain shifts by emphasizing underperforming or rare data domains [420], [278]; (4) *Model-based optimization*, which builds predictive models to map data mixing ratios to loss and task performance. Approaches include linear predictive modeling (e.g., REGMIX [263]), nonlinear function fitting [152], [439], [160], scaling law-based estimation [323], and latent source attribution [251].

• *Data Synthesis.* We introduce data synthesis techniques designed to address the following key challenges: (1) *Mitigating harmful characteristics* such as toxicity or bias, which can be inherited or amplified in synthetic data (e.g., program-aided verification [496], semantic scoring [173], and multi-agent consistency filtering [346]); (2) *Balancing data utility and privacy*, through privacy-preserving synthetic rewriting and key-entity obfuscation methods during the RAG stage [450]; (3) *Generating diverse and logically consistent reasoning data* using approaches like formal proof-based validation [178], Chain-of-Thought (CoT) branching and error correction [173], and high-quality problem synthesis guided by structure and complexity constraints [260], [442]; (4) *Automating human-like evaluation and feedback generation* with LLM-based preference modeling [71], judge models for response ranking [476], and clustering-based diversity quantification [92].

• *Data Pipelines.* We first introduce *frameworks* that integrate basic data processing operators and interfaces, serving as the general foundation for building data pipelines [90], [305], [368]. Then we showcase *typical pipelines* with heuristic mechanisms that properly arrange these operators (mainly for LLM pre-training) [311], [236], [310]. Finally, we discuss strategies that go beyond heuristic designs to further optimize these data processing pipelines [91].

Data Storage for LLMs (§ 2.4). We review data storage techniques for LLMs from the following main aspects.

• *Data Formats.* We review commonly-used dataset and model data formats for LLMs. Dataset formats include *TFRecord* [44], *MindRecord* [40] for multimodal data, and *tf.data.Dataset* that can be directly fed into LLMs [43]. For model data storage, there are formats like *Pickle* [13] and *ONNX* [27].

• *LLM Data Distribution.* LLM data distribution aims to store data across multiple storage nodes in a cluster, which mainly serves for storing large-scale LLM training data. Key approaches include (1) distributed storage systems like JuiceFS [16] and 3FS [15]; and (2) heterogeneous storage systems for model data (e.g., across GPUs and CPUs) [333], [334], [337], [336], [435].

• *LLM Data Organization.* LLM data organization aims to transform data into a format suitable for storage and retrieval (mainly for the RAG stage) in heterogeneous forms. First, for vector RAG, relevant techniques include content formatting [97], [172], [57], [89], chunking [480], embedding [94], [24], [249], compression [50], [380], [381], [381]. Second, for graph RAG, we discuss indexing techniques such as generating textual summary for quick retrieval [127], [164], [136]. We also introduce the systems that integrate these techniques, including vector search engines [125], [26], [34], [25] and graph storage platforms [292], [65], [1].

• *LLM Data Movement.* LLM data movement aims to improve the speed of data movement across storage and compute nodes. Relevant techniques include (1) caching data [219], [161], [469]; (2) offloading data/operator to multiple devices (e.g., across CPUs) [158], [67], [159], [468]; and (3) overlapping of storage and computing in training stage [466], [479].

• *LLM Model Data Fault Tolerance.* LLM model data fault tolerance aims to enhance the ability to recover from system failures during model training. Relevant techniques include (1) checkpointing [291], [194], [403], [389], which stores checkpoints across a hierarchical storage system; and (2) redundant computation, which leverages redundant states of LLM in parallel training (e.g., pipeline parallelism [382], hybrid parallelism [186], [147]) to support rapid fault recovery.

• *KV Cache in LLMs.* KV caching in LLMs is essential for enabling fast and efficient inference by managing key-value memory usage. Existing techniques include: (1) *Memory layout and allocation*, which optimize the physical organization of KV memory for high performance and scalability [220], [428]; (2) *Storage offloading*, which places KV data on suitable storage media to balance speed and capacity [197], [148]; (3) *KV compression*, which reduces memory footprint through techniques like encoding compression [265], [255], [150]; (4) *Efficient indexing*, which accelerates KV access via specialized retrieval structures [440], [478].

Data Serving for LLMs (§ 2.5). We provide an overview of data serving techniques tailored for LLMs from four aspects.

• *LLM Data Shuffling.* LLM data shuffling aims to determine the appropriate order of data application during stages like LLM training and RAG. In the training stage, we discuss data pruning techniques (e.g., sample-scoring-based approaches [137], [66], model-state-based approaches [372], [56], [416], [276]) and data-centric training strategies [123]. In the RAG stage, we discuss RAG knowledge filtering [280], [114], [87] and re-ranking [128], [12], [318], [47].

• *LLM Data Compression.* LLM data compression aims to compress the model's input data to stay within the context window limit or to facilitate model understanding. Relevant techniques include: (1) RAG knowledge compression (e.g., rule-based [427], [348], [200] and model-based method [101], [335]); and (2) prompt compression (e.g., metric-based [189], [190] and model-based method [303], [293], [102]).

• *LLM Training Data Packing.* LLM training data packing aims to ensure uniform sequence lengths in training inputs. Relevant techniques include: (1) short sequence insertion [116], [259]; (2) optimizing sequence combination [218], [316]; and (3) semantic-cased packing [364], [349]).

• *LLM Inference Data Provenance.* LLM inference data provenance aims to ensure the factual consistency of LLM-generated content. Relevant techniques include: (1) embedding markers [482], [105], [256]; and (2) statistical provenance [212]).

1.2 Techniques of LLM4DATA

LLM for Data Manipulation (§ 3.1). LLMs have been increasingly applied to data manipulation tasks, with the goal of preparing high-quality datasets for non-LLM applications and enhancing data quality for downstream usage. Key areas include data cleaning, data integration, and data discovery.

• *Data Cleaning.* This task involves standardizing and refining datasets through a series of operations. We highlight three major subtasks: (1) Data Standardization, which reformats data samples using handcrafted standardization prompts [279], [63] or agents that generate cleaning operations or pipelines [319], [237]; (2) Data Error Processing, which identifies and corrects noisy data via direct LLM prompting [103], [461], [432], context-enrichment techniques [78], [74], or task-specific fine-tuning for error handling [432]; (3) Data Imputation, which fills in missing values using explicit imputation instructions and retrieval-augmented generation (RAG) methods [129].

• *Data Integration.* This task focuses on identifying and reconciling semantically related datasets across heterogeneous sources. We review two core subtasks: (1) Entity Matching, which aligns data entries referring to the same real-world entity using structured prompts [308], [134], sometimes augmented with predefined code-based reasoning strategies [430]; (2) Schema Matching, which establishes correspondences between schema elements using direct prompting [304], RAG techniques incorporating multiple models [267], knowledge graph-based methods [277], and agent-based workflow generation [320], [340].

• *Data Discovery.* This task aims to extract informative insights from a dataset. We cover two key subtasks: (1) Data Profiling, which generates descriptive metadata and summaries using task-specific prompts [456], [58], and enhanced with context via RAG techniques [72]; (2) Data Annotation, which assigns semantic labels or types through various prompting strategies [203], [204], [217], supported by classical retrieval-based [408] and LLM-generated context [163].

LLM for Data Analysis (§ 3.2). LLMs significantly improve the analytical capabilities across structured, semi-structured, and unstructured data.

• *Structured Data Analysis.* For relational data analysis, natural language interfaces allow users to write high-level questions instead of SQL/Python code [452]. Multi-step QA frameworks (e.g., TAPERA [475] and ReAcTable [464]) decompose complex queries, while some end-to-end solutions fine-tune LLMs specifically for tabular tasks (e.g., TableGPT [240]), apply content retrieval (e.g., CABINET [306]) or convert tables into images for analysis (e.g., Table-LLaVA [477]). For graph data, LLMs facilitate semantic queries with GQL generation (e.g., R^3-NL2GQL [493]) and knowledge-aware QA by retrieving or reasoning over relevant subgraphs [424].

• *Semi-Structured Data Analysis.* Meanwhile, handling semi-structured data (e.g., JSON and spreadsheets) remains challenging. Recent benchmarks (e.g., TEMPTABQA [165] and SPREADSHEETBENCH [281]) reveal substantial performance gaps.

• *Unstructured Data Analysis.* Finally, unstructured data analysis leverages LLMs to address document and program analysis tasks. For document analysis, OCR-dependent approaches involve performing OCR on document images followed by the integration of textual, layout, and visual features for reasoning (e.g., UDOP [376] and DocFormerV2 [62]). OCR-free methods directly generate the answer with end-to-end multimodal LLMs (e.g., Pix2Struct [225] and DUBLIN [49]). For program analysis, LLMs could serve as vulnerability detection tools using program analysis based training (e.g., PDBER [271]) or case-driven prompt engineering (e.g., VUL-GPT [270]). For program related analysis, LLMs could summarize repositories (e.g., SCLA [284]) or serve as a repository-level code completer (e.g., RepoFusion [357]) using their powerful semantic reasoning abilities.

LLM for Data System Optimization (§ 3.3). LLMs equipped with advanced reasoning and code generation capabilities have been increasingly adopted in core system optimization tasks. These include: (1) configuration tuning (identifying optimal system settings); (2) query optimization (rewriting or refining input queries for performance gains); and (3) anomaly diagnosis (analyzing system issues to ensure performance reliability).

• *Configuration Tuning.* This task leverages LLMs to determine effective configuration parameters for improved system performance through: (1) Prompt engineering tailored to tuning tasks, using both manually crafted [243], [132], [156] and automatically generated prompts [491], [473]; (2) Retrieval-augmented generation (RAG), which incorporates prior tuning experiences during offline knowledge base preparation [223] and online knowledge retrieval [96]; (3) Objective-aligned tuning, which is enhanced through targeted training techniques [491], [177].

• *Query Optimization.* This task utilizes LLMs to rewrite queries or improve execution plans by: (1) Designing optimization-oriented prompts that include explicit guid-

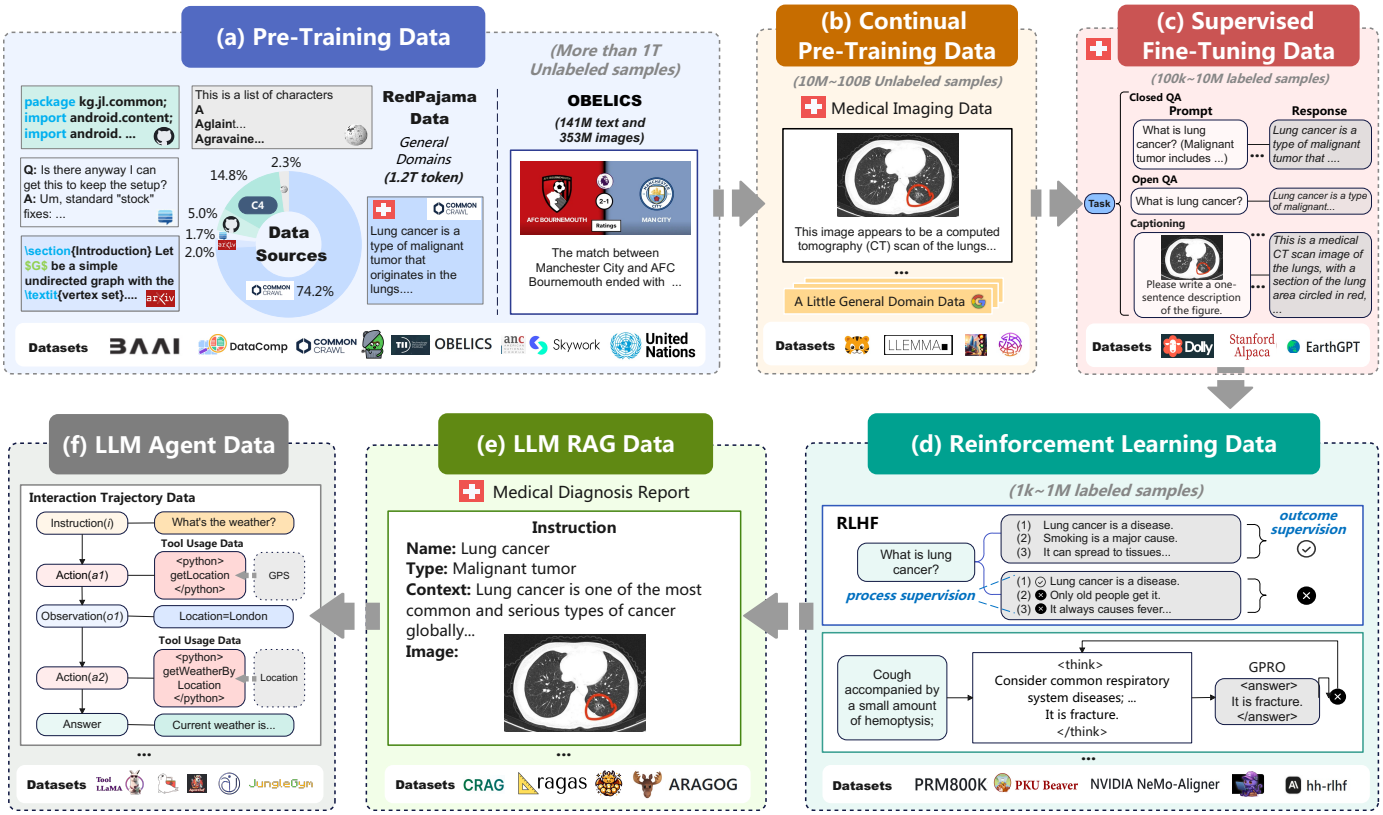

Fig. 2: **Example Data Characteristics across LLM Stages** - (a) Pretraining data [109], [224], (b) Continual pre-training [111], (c) SFT [447], (d) Reinforcement learning [429], [162], [253], (e) RAG [415], (f) Agent [396], [351].

ance [363], [491], [438] and in-context examples [248]; (2) Enriching optimization knowledge using RAG techniques, including LLM-generated and hybrid retrieval strategies [369]; (3) Enhancing optimization performance through task-specific training [53], [196], [438].

• *Anomaly Diagnosis.* This task involves identifying the root causes of anomalies and suggesting effective solutions via: (1) Direct LLM prompting based on detailed diagnosis context [155]; (2) RAG-based enrichment using relevant historical diagnosis experience [490], [425]; (3) Multi-agent collaboration mechanisms for comprehensive diagnosis [490], [359].

1.3 Comparison with Existing Surveys

Different from existing LLM and data management surveys [405], [55], [86], [398], [272], [274], [374], [488], our survey offers a comprehensive and detailed overview of the key intersections between LLMs and data management, highlighting how they can mutually benefit from each other. We uniquely position our work at the intersection of data for LLMs (e.g., how to acquire, process, store, and serve LLM data) and LLMs for data (e.g., how LLMs can be leveraged to enhance data management tasks).

• *We propose the **IaaS** concept as a principled lens to assess LLM dataset quality.* The IaaS concept identifies four essential dimensions, including inclusiveness, abundance, articulation, and sanitization. This concept is promising to offers an evaluative criteria for guiding data management and understanding its impact across the LLM development lifecycle (see Section 2.1).

• *We investigate the unique characteristics of data across different LLM development stages* (Figure 2), and provide a systematic overview of the associated challenges and techniques in data processing, storage, and serving (Table 1). In contrast, prior surveys [405], [55], [86] primarily center on the pre-training stage without covering the full LLM lifecycle like supervised fine-tuning (SFT), retrieval-augmented generation (RAG), and agent-based applications.

• *We provide a lifecycle-based taxonomy of DATA4LLM, introducing key tasks in data processing, storage, and serving.* For each task, we summarize representative methodologies, discuss their design principles, and analyze their strengths and limitations. In comparison, [405] focuses on deduplication and filtering, [55] emphasizes data selection, and [373] reviews data annotation strategies, none of which offer a systematic perspective across the data management pipeline.

• We *introduce recent advances in LLM4DATA, outlining key components of LLM-driven data optimization.* While earlier work [488] has investigated the application of classical machine learning in data management, it largely neglects the distinctive strengths and limitations of LLMs, particularly in manipulating data for non-LLM tasks, processing semi-structured and unstructured data, and enabling system-level optimizations.

• *We highlight open challenges and future directions* from both ends: (1) improving data management techniques to meet practical LLM training and deployment needs (e.g., efficient data evaluation, scalable multi-modal storage), and (2) enhancing LLMs' ability (e.g., private knowledge understanding, informative representation for non-sequential and non-textual data) to perform complex data management tasks across diverse real-world scenarios.

2 Data Management for LLM (DATA4LLM)

2.1 "IaaS" Concept of LLM Data

Based on our investigation of over 400 papers [2], we introduce the **IaaS** concept for evaluating the quality of LLM datasets.
(1) Inclusiveness: LLMs require data with broad and diverse coverage across multiple dimensions, including domains (e.g., general knowledge, specialized fields like finance, medicine, math [98], and physics [233]), task types (e.g., question answering, summarization, code completion [401], [290], [353], [45], [436]), data sources (e.g., GitHub, Wikipedia [149], [11], [330], [347]), languages [93], [347], expression styles (e.g., academic, casual, formal [282], [470]), and data modalities (e.g., text [149], [11], images [145], [185], videos [437], [216], [390], tables [330]).
(2) abundance: LLMs require data with appropriate volume and balanced composition to prevent overfitting on homogeneous data. Specifically, abundance of data involves: (*i*) constructing well-balanced datasets during pre-training [139], [302], [420], [263], (*ii*) adjusting data ratios to align with target applications during fine-tuning [278], [135], and (*iii*) continually enhancing domain-specific capabilities while maintaining acceptable general performance degradation in continual pre-training [323], [160]. Notably, the strength of LLMs lies not only in large-scale data [282], [481], [11], [330], [149], [347], but also in constructing purposefully balanced datasets, which can further accelerate training and reduce computational cost.
(3) articulation: LLMs require data that exhibit strong articulation, including three key aspects: (*i*) the data should be well-formatted (e.g., proper punctuation and capitalization [90]), clean (free from duplicates, typos, and irrelevant content such as spam or gibberish [90]), and self-contained, featuring clear, fluent, and unambiguous language [282], [470], (*ii*) the data should be instructive [178], [179], [98], i.e., offering sufficient context, guidance, and intermediate explanations that help the model connect questions to relevant background knowledge and understand the reasoning process. (*iii*) the data should involve step-by-step reasoning[230], [442], [346], [173], [496], such that enhancing the LLMs' reasoning capabilities by decomposing complex tasks into smaller, interpretable steps.
(4) Sanitization: LLMs require data to be *sanitized*, meaning it is rigorously controlled and filtered to remove harmful elements while maintaining inclusiveness and neutrality. This involves four critical dimensions: (*i*) *Privacy compliance*, which requires the exclusion of personally identifiable information (e.g., ID numbers, phone numbers), inferred social relationships, and geolocation-related metadata [450], [268], [275]; (*ii*) *Toxicity-free content*, ensuring the complete removal of hate speech, incitement to violence, and psychologically harmful language, as well as eliminating any discriminatory or aggressive semantic constructs [296]; (*iii*) *Ethical consistency*, which prohibits the presence of extremist ideologies, instructions for illegal activities, and stereotype-reinforcing narratives that may cause social harm [345], [360], [296]; and (*iv*) *Risk mitigation*, filtering out unverified medical claims, politically misleading information, and culturally insensitive expressions to prevent misinformation and value misalignment. Sanitized data must maintain a neutral tone and adopt an inclusive contextual framework, serving as a critical foundation for building safe LLMs [345], [360].

2. https://github.com/weAIDB/awesome-data-llm

2.2 Data Characters across LLM Stages

Next we specifically discuss the data characteristics across different LLM stages, together with the distinct techniques for data processing, storage, and serving (Table 1).

Data for Pretraining. In the pre-training stage, LLMs rely on TB-scale, diverse datasets to acquire broad language and even cross-modality understanding capabilities, while reducing the risk of overfitting. These datasets are typically sourced from a wide range of domains and formats, including web crawls (e.g., HTML pages and WARC files [11]), open-source code repositories (e.g., raw source code files with metadata [14]), books (e.g., plain text or EPUB formats [497]), academic papers (e.g., LaTeX source or PDF-converted text [2]), and interleaved image-text corpora (e.g., aligned captioned images in JSON or WebDataset format [224]).

Data for Continual Pre-training. Continual pre-training (or continued pre-training) typically involves datasets containing millions to billions of tokens, which are often over 100 times smaller than those used in the initial pre-training stage. The primary objective is to fill knowledge gaps and adapt the model to specific domains. Representative domain-specific datasets are like: (1) Finance: BBT-FinCorpus [273], a large-scale and diverse financial datasets comprising approximately 300 GB of text; and (2) Healthcare: Medical-pt [429], a Chinese-English medical dataset containing 360,000 entries curated from medical encyclopedias.

Data for Supervised Fine-Tuning (SFT). Unlike pre-training, SFT relies on data presented in the form of instruction-response pairs, where the response includes not only the correct answer but also guidelines on tone, style, and reasoning steps to ensure user-friendly output.

The SFT stage typically involves much smaller datasets compared to pre-training. These datasets often consist of thousands to millions of labeled examples, with each example carefully crafted to guide the model in learning a specific, narrower set of tasks. For instance, in Figure 2, (1) the summarization task constructs prompts using *problem descriptions and summarization objects*; (2) closed QA using *questions and corresponding knowledge texts*; (3) open QA tasks using *only questions* without knowledge text; and (4) captioning tasks using *task descriptions and images*. These prompts are paired with unique responses for model finetuning.

The composition of SFT datasets varies based on the application scenarios:
(1) General Instruction Following: For LLMs as general-purpose chatbots, SFT data include instructions for various daily tasks. Databricks-dolly-15K [110] is a corpus containing over 15,000 records. It encompasses seven types of tasks, including creative writing, closed QA, open QA, summarization, information extraction, classification, brainstorming. This dataset is designed to enhance LLM to better adapt to specialized outputs that align with human-style requirements across diverse tasks. For example, in text summarization, it provides concise summary statements; whereas in text organization tasks, it structures outputs in table-of-contents format.
(2) Specific Domain Usage: For models specialized in fields such as law, finance, or medicine, the SFT data focuses

TABLE 1: **Technique Comparison** - Data Processing, Storage, and Serving Techniques for Different LLM Stages. "N/A" indicates that no relevant work has been reported yet, although the corresponding techniques could potentially be applied.

Stage		Pre-training / Incremental Pre-training	Supervised Fine-Tuning	Reinforcement Learning	Inference	RAG	Evaluation
Data Processing	Acquisition	✓	✓	✓	N/A	✓	✓
	De-duplication	✓	✓	N/A	N/A	N/A	N/A
	Filtering	✓	✓	N/A	N/A	×	N/A
	Selection	✓	✓	N/A	N/A	N/A	N/A
	Mixing	✓	✓	×	N/A	×	×
	Synthesis	✓	✓	✓	N/A	✓	✓
Data Storage	Distribution	Distributed File System Model Offload (GPUs, CPUs)	Model Offload (GPUs, CPUs)	Model Offload (GPUs, CPUs)	Model Offload (GPUs, CPUs)	Model Offload (GPUs, CPUs)	Model Offload (GPUs, CPUs)
	Transmission	Caching Data Placement Parallelized Pipeline Data/Operator Offloading (CPUs)	Parallelized Pipeline Data/Operator Offloading (CPUs)	Parallelized Pipeline Data/Operator Offloading (CPUs)	×	N/A	N/A
	Fault Tolerance	✓	✓	✓	×	×	×
	KV Cache	N/A	N/A	N/A	Cache Space Management KV Indexing KV Placement KV Shrinking	KV Placement KV Shrinking	N/A
Data Serving	Selection	Sample-Scoring-Based Model-State-Based	Model-State-Based Experience-Based	N/A	×	SLM-Based Filtering LLM-Based Filtering Metric-Based Re-ranking LLM-Based Re-ranking	×
	Compression	N/A	N/A	N/A	✓	✓	N/A
	Packing	✓	✓	✓	×	×	×
	Provenance	×	×	×	✓	N/A	×

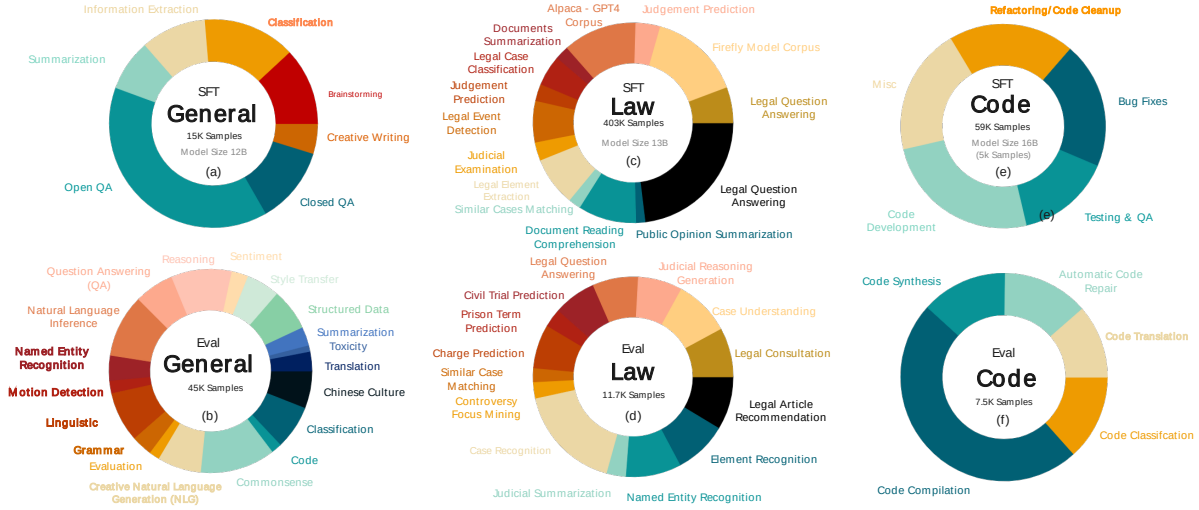

Fig. 3: **Example LLM Data Distributions** - (a) General Domain (SFT)[110], (b) General Domain (Eval) [244], (c) Law (SFT)[447], (d) Law (Eval)[115], (e) Code (SFT) [294], (f) Code (Eval)[208].

on tasks pertinent to these fields. For example, DISC-Law-SFT [447] is a legal SFT dataset containing 295k data entries from various legal scenarios, such as legal information extraction (32k), legal judgment prediction (16k), legal event detection (27k), and legal question-answering (93k). Similarly, Medical-SFT [429] is a medical SFT dataset (totaling 2,060k pieces), composed of medical inquikry data (790k), online medical encyclopedia QA data (360k), English medical inquiry data (110k), medical knowledge graph QA data (79k). For tasks such as legal question-answering and legal judgment prediction, the data is structured as triplets, comprising the prompt, response, and supporting reference information (e.g., legal provisions, case-based evidence, or regulatory documents). For the remaining tasks, they all take the form of instruction pairs composed of prompt and response.

Data for Reinforcement Learning (RL). RL is generally divided into two types: one is RLHF (Reinforcement Learning with Human Feedback), and the other is Reasoning-oriented Reinforcement Learning (RoRL).

(1) RLHF: RLHF data is typically smaller than SFT data (e.g., thousands to dozens of millions of data samples), which involve more complex data annotations. Specifically, annotators compare multiple candidate responses to the same instruction and rank them according to human preference (e.g., levels from most helpful to least helpful). Collecting these preference pairs or rankings is more time-consuming than constructing instruction-response pairs in SFT.

In the general domain, UltraFeedback [113] consists of 64,000 samples. For each sample, different models are used to generate 4 responses for each prompt (totaling 256,000 responses). GPT-4 is then employed to generate feedback for these four responses, which is used to help LLMs to generate outputs that are in line with human standards and appropriateness.

In specific domains such as healthcare, Medical-RLHF [429] has 4,000 random questions from a Chinese medical dialogue dataset. Each question is paired with a well-organized answer (i.e., the human doctor's reply) and a weaker answer from Llama-based model fine-tuned over synthesized QA samples. These labeled data are used to train a reward model. During the training of the LLM, the reward model provides feedback based on the LLM's answers, guiding the

training process towards generating high-quality responses.

(2) RoRL: Compared to the complex annotated data in RLHF, RoRL allows the model to discover the best reasoning approach on its own through the correctness of the reward model. Specifically, it focuses on tasks requiring long-term reasoning, such as mathematical, coding, and logical designing experiments [162]. Under the premise of providing feedback on whether the answer is correct or not, algorithm such as the Group Relative Policy Optimization (GRPO) [162] and long-CoT RL [377] are adopted to train the model to independently discover the optimal problem-solving steps and converge.

Data for Retrieval-Augmented Generation (RAG). The RAG stage differs from above training stages, which involves large-scale dataset (reference corpus) for LLMs to retrieve from during inference. In this stage, data must be strictly reviewed to ensure authenticity and validity, while dynamic data requires real-time updates. The domain of RAG datasets varies depending on the specific application scenarios. For instance, (1) in the medicine-specific LLM application (Medical-Graph-RAG), MIMIC-IV is used as the RAG dataset [415]. This dataset contains data from over 65,000 ICU patients and more than 200,000 patients treated in emergency departments; (2) in the legal field, the RAG knowledge base used by DISC-LawLLM [447] contains more than 800 national and local laws, regulations, and rules, as well as 24,000 legal-related exam questions. Besides, RAG data can include users' historical conversation records or personal information, in order to build a user-personalized LLM [350], [451], [453].

Data for LLM Evaluation. Suitable evaluation datasets are essential for evaluating the performance of LLMs. They provide representative data samples that reflect different aspects of an LLM's capabilities.

In the general domain, the MMMU benchmark is used to assess the performance of LLMs across major multi-modal tasks in six key disciplines, covering 30 subjects and 183 subfields. It is built from 11,500 carefully curated questions and effectively tests models' perception, knowledge, and reasoning abilities [448].

In specific domains, typical evaluation datasets include those in coding, healthcare and law domains: (1) OpenAI's HumanEval dataset includes 164 programming problems, complete with function signatures, docstrings, bodies, and multiple unit tests. These problems are handcrafted to ensure they are not part of the training sets used for code generation models [95]; (2) MedQA [198] contains a large number of medical exam questions from various regions, totaling 61,097 questions; (3) LexEval [232] constructs 23 evaluation tasks based on a legal cognitive classification framework, covering different aspects of legal knowledge, with at least 100 evaluation samples for each task.

Data for LLM Agents. Beyond vanilla LLMs, agents strive for more advanced capabilities such as planning, tool orchestration and multi-turn dialogue capability [262]. These capabilities impose higher requirements on the training data for LLMs. First, many studies [396] aim to enhance planning abilities through interaction trajectory data, which refers to a sequence of records generated during the interaction between the agent and the environment, typically represented as (instruction i, action a_1, observation $o_1, \ldots,$ action a_n). UltraInteract [446] takes the instruction as the root node, and uses both the correct actions and their corresponding incorrect actions as nodes to construct a preference trajectory tree, enabling the agent to learn the human preference of different actions. Second, other studies focus on enhancing the agent's tool usage capabilities using tool usage data. For instance, AutoTools [351] fine-tunes models on tool data that is labeled with special tags, such as `<python>`*code*`</python>`, thereby grounding language in concrete tool invocations. Third, to enhance the agent's multi-turn dialogue capability, UltraChat [117] employs an additional LLM to simulate user instructions and conversational content, thereby collecting multi-turn dialogue data.

TABLE 2: Data Acquisition for LLMs.

Method	Objective	Solution	Tools
Website Crawling	HTML Textual Content Extraction	Rule-based	Trafilatura [73]
		Rule-based	BET [144]
		ML-based	Dragnet [313]
	Automate Browser Interactions	HTML parsing	Beautiful Soup [6]
		Control web driver	Selenium [19]
		Wrap high-level API	Playwright [30]
		DevTools protocol	Puppeteer [31]
Layout-based	Content Extraction from Handwritten or Non-text Data	Model pipeline	PaddleOCR
		Model pipeline	MinerU [392]
		Multimodal LLM	GOT2.0 [407]
		Multimodal LLM	Fox [257]
Entity recognition & linking	New Sample Derivation	Bi-Transformer	ReFinED [68]
	Translation Consistency	Seq2seq Framework using References	AACTRANS [215]
	Text-Image Integration	Multimodal LLM	UMIE [367]

2.3 Data Processing for LLM

2.3.1 Data Acquisition

Unlike classic machine learning, which primarily relies on collecting labeled data within a specific domain for supervised training (e.g., data for sentiment analysis and sentence similarity estimation), data acquisition for LLMs typically (1) relies on large-scale web scraping to collect extensive data across diverse domains for unsupervised pretraining and (2) employs techniques such as layout analysis and entity linking to extract additional data from the collected content.

Principles

Unlike classic ML data acquisition, LLMs rely heavily on large-scale web scraping to ensure broad coverage and robust generalization. The main challenge is extracting high-quality textual content, often aided by layout-based and entity-linking methods. Managing time and resource efficiency at scale remains vital.

Data Sources. The data is gathered from two primary sources:

(1) Public Data, often freely available under open licenses, include resources such as webpages [11], books [497], and publicly accessible code repositories [214].

- *Webpage sources* provide extensive pre-processed website content, such as 1.56T english text from crawled websites in C4 [331], 6.6B multilingual pages in mC4 [431], 6.3 trillion tokens of multilingual pages in CulturaX [297].
- *Digitized books* supply structured, high-quality text, such as over 75,000 eBooks in Project Gutenberg [38], over two

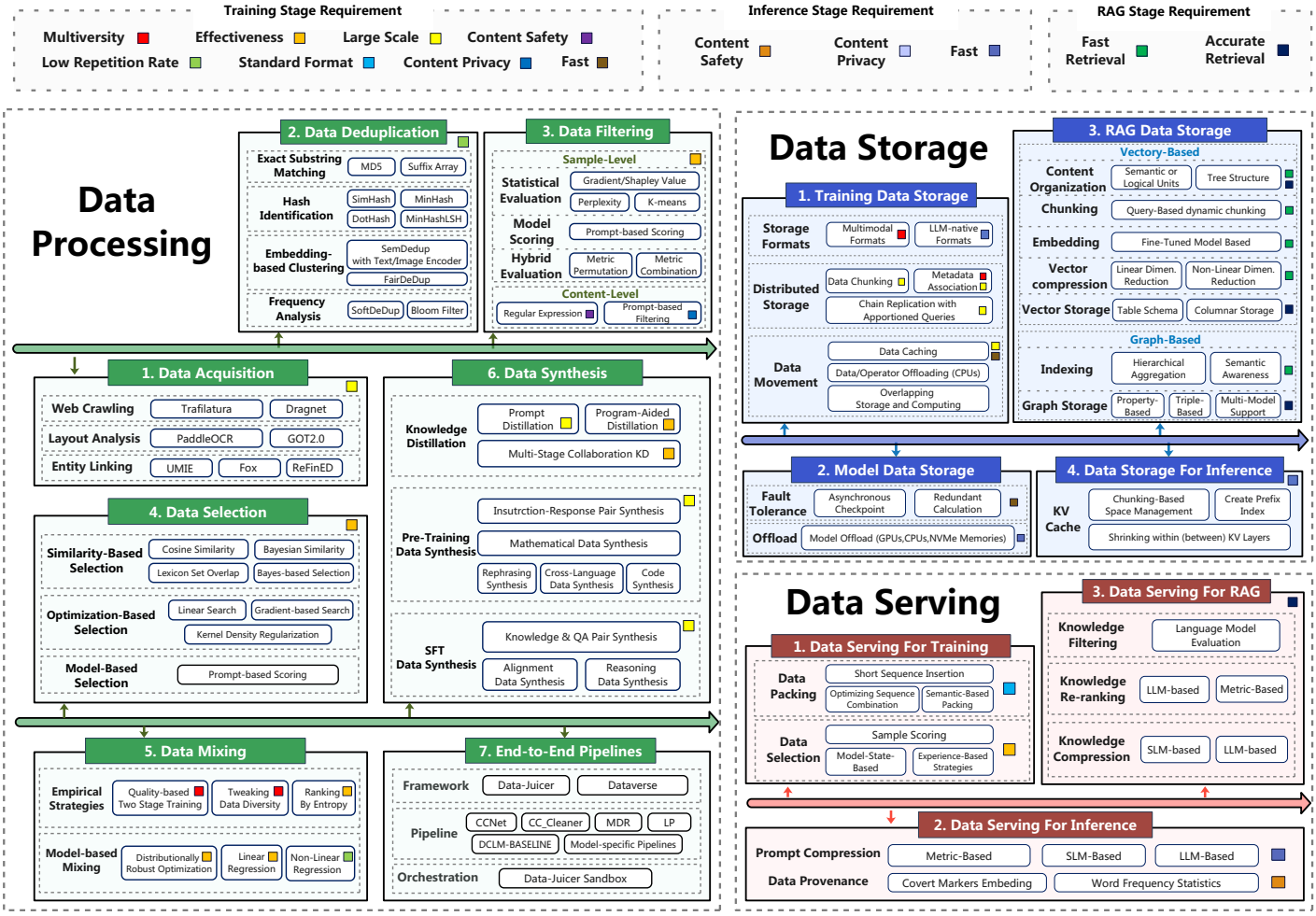

Fig. 4: Overview of DATA4LLM Techniques.

million free ebooks in Open Library [28], and film-aigned book descriptions in BookCorpus [497]).

• *Code repositories* (e.g., GitHub [14], GitLab [20], Bitbucket [7]) offer abundant programming data that can facilitate code search and analysis tasks, such as CodeSearchNet [181] with 2M (comment, code) pairs.

(2) Private Data involve proprietary or confidential information not publicly available, such as internal company documents, customer support logs, application event logs, subscriber-only content (e.g., premium news articles, licensed scientific databases). Collecting this data requires careful attention to ethical and legal constraints (e.g., GDPR, CCPA) and mandates removing sensitive details (e.g., employing anonymization or pseudonymization) and using secure pipelines (e.g., CI/CD systems) with encryption and role-based access controls. For instance, proprietary codebases and user-generated content (chat logs, Q&A sessions) must be gathered under secure processes to maintain confidentiality.

Data Acquisition Methods. As shown in Table 2, there are three main techniques for data acquisition, including website crawling, layout analysis, and entity recognition and linking.

(1) Website Crawling. Most data are obtained through website crawling, which aims to extract textual content from crawled HTML files or multimodal image-text pairs using various extraction tools and browser automation assistants.

Generally, we first parse the raw HTML to separate meaningful textual content from boilerplate elements. Second, since typical extraneous components (e.g., headers, footers, advertisements, sidebars) often contribute little to the data value (e.g., for LLM training), we execute scripts (using CSS selectors or XPath queries) to identify and extract critical elements like article text, headlines, dates, and author bylines. Third, once the relevant text has been scraped, we store it in structured format such as JSON, CSV, database (see data storage in Section 2.4) for further processing. Specifically, for image elements encountered in HTML files, the image source URL is recorded, and the content of the `alt` attribute within the `<img>` tag is extracted and utilized as the corresponding image's textual caption.

• *Rule-based Crawling.* Most existing tools use heuristic rule-based matching algorithm. Trafilatura [73] is a heuristic algorithm based on hand-crafted rules (e.g., match HTML DOM nodes with the class equal to "navbar" to filter the navigation bar). BET [144] employs the cumulative HTML tag distribution to find the largest region of fewest tags per text and extracts the corresponding text as the main content.

• *ML-based Crawling.* Since many website regions cannot be easily classified by rules, some works [76], [73] design a HTML tag classifier to judge whether a DOM node contains textual content, where they adopt L^2 regularized logistic regression that inputs text density features and word frequencies in "id" and "class" attributes and outputs the probability that a given

node contains textual useful content.

• *Auxiliary Tools.* Moreover, some auxiliary tools integrate user-friendly APIs for operating and interacting with HTML DOM trees. Beautiful Soup [6] is widely used to parse the raw HTML in Python. Selenium [19] automates browser actions and handles dynamic pages by controlling a web driver that communicates with the browser. Playwright [30] provides a high-level API to automate browser tasks while Puppeteer [31] communicates directly with the browser using the DevTools Protocol, allowing for headless browser interactions (e.g., in JavaScript-heavy websites).

(2) Layout Analysis. Layout analysis focuses on extracting textual content from handwritten or non-textual data (e.g., from the crawled ones), which can contain valuable information and require advanced layout analysis techniques for effective extraction. Existing methods include pipeline-based and end-to-end approaches.

• *Layout Analysis Pipelines.* Intuitively, many works adopt OCR technology (e.g., Tesseract [202]) to convert raw data (e.g., scanned books) into machine-readable formats [18], [392] in a pipeline manner, which consist of multiple small models. PaddleOCR [18] passes an image through a Layout Analysis model, which divides the image into different regions such as text, tables, and formulas for separate processing. The table area is sent to the Form Recognition module for structured recognition, and the text areas and formulas are input to the OCR engine for text recognition. Finally, the Layout Restoration module reconstructs all the regions in textual format using heuristic rules based on the relative location information of different extracted regions.

Similarly, MinerU [392] works in a pipeline manner. It fine-tunes LayoutLMv3 [180] for layout detection and YOLOv8 [391] for formula detection to improve the system's generalization (handling a wider range of document types). The detected data are kept in markdown or JSON format.

• *End-to-End Models.* End-to-End layout analysis refers to adopt multi-modal LLMs to conduct end-to-end text acquisition. For instance, GOT2.0 [407] is a acquisition model composed of (*i*) a high-compression encoder that transforms the image to tokens, (*ii*) a long-context decoder that outputs the corresponding OCR results, and (*iii*) a linear layer acting as the connector to map the channel dimension between the vision encoder and the language decoder. Another example is Fox [257], which employs the natural content-aware CLIP-ViT [326] and the artificial content-aware Vary [406] as two vision encoders, enabling the model to perform fine-grained interactions and multi-page document understanding. The end-to-end architecture reduces maintenance costs and enhances versatility, enabling the recognition of more complex elements (e.g., charts, sheet music) and supporting improved readability formats for formulas and tables (e.g., LaTeX, Markdown). However, due to the use of LLMs with larger parameter size (e.g, <20M for PaddleOCR vs. 580M for GOT2.0 and 1.8B for Fox), the inference efficiency of these methods still needs improvement.

(3) Entity Recognition & ***Linking.*** Additionally, we can derive more valuable LLM samples by identifying and linking entities from the above extracted data. WEBIE [412] introduces a large-scale, entity-linked information extraction dataset with 1.6M sentences from Common Crawl. It links entities using ReFinED [68], and applies distant supervision (DS) to extract 4.8M triples, where each triple consists of a subject, a relationship, and an object.

Furthermore, to ensure the consistency of derived and origin samples (e.g., translation across English and other languages), Alignment-Augmented Consistent Translation (AACTRANS) model [215] uses a Seq2Seq framework that incorporates reference text in the target language to guide translations, ensuring consistency across related pieces of text. During training, aligned text pairs are augmented with reference-based word alignments to bias the model toward consistent translations. At inference, a common reference translation of the original sentence is used to align and translate related extractions using the AACTRANS model.

However, AACTRANS fails to leverage shared knowledge across tasks, limiting the alignment performance. Instead, UMIE [367] integrates text and visual inputs and produces structured outputs to learn linking knowledge from multiple tasks. The UMIE model is composed of four modules: (1) a text encoder for task instruction comprehension, (2) a visual encoder for image understanding, (3) a gated attention mechanism for cross-modal integration, and (4) a text decoder for structured output generation. Following different task instructors, UMIE is capable of performing various MIE tasks and generating corresponding structured outputs, thereby facilitating knowledge sharing.

Notably, recent LLMs could automatically learn the relationships among samples from randomly provided data, rendering the explicit entity linking an optional procedure in the data acquisition process [119].

TABLE 3: Data Deduplication for LLMs.

Method	Objective	Modality	Work
Exact substring matching	Deduplicate samples with identical substrings	Text	MD5 [122] Suffix Array [299]
Hashing identification	Deduplicate samples with similar substrings	Text	SimHash [88] MinHash [81], [122], [299] MinHashLSH [347], [358] MinHash + Bloom Filter [207] DotHash [298]
Frequency analysis	Down-weighing samples with higher commonness	Text	SoftDeDup [167]
Embedding-based clustering	Deduplicate samples with identical topics but different formats	Text + Image	SemDeDup [46] SemDeDup + SSL Prototypes[385] FairDeDup [360]

2.3.2 Data Deduplication

The collected raw data often contains significant redundancy, which can negatively impact LLM performance either by reducing its generalization ability to new or rarely-seen tasks [299] or by memorizing and overfitting to the repeated subsets [169], [422]. Various deduplication methods have been proposed to detect and mitigate duplication, either by (1) completely removing duplicate samples [122], [299], [347], [358], [207], [46], [385], [360] or by (2) down-weighing duplicate samples for data resampling [167]. We classify these methods into four main categories.

Exact Substring Matching. Exact substring matching methods identify and remove exactly identical samples across datasets, which can happen if (1) a sample references another sample (e.g., a report related to another), or (2) two individual datasets accidentally include the same sample (e.g., a webpage

of a popular website). It is commonly used as a preliminary step to remove duplications. Relevant methods leverage techniques like hashing [122] and suffix array [299] at the sample or sentence level.

Principles

Compared to structured classic ML data, LLM data is unstructured and requires careful identification and removal of duplicate or near-duplicate content from training datasets to improve efficiency, prevent overfitting, and mitigate bias using statistical metrics like perplexity or model evaluation. Challenges include (1) how to encode semantic texts into representations that could be precisely and efficiently compared and (2) the scalability of the deduplication methods.

• *Sample-Level.* [122] conducts sample-level deduplication by calculating the MD5 hashing value of each sample and deduplicate samples with identical MD5 values.

• *Sentence-Level.* [299] performs sentence-level deduplication by using Suffix Array, which combines all the samples into one sentence, computes the sentence Suffix Array, and deduplicates samples with common prefixes in the Suffix Array. Suffix Array [283] is a data structure that stores the starting indices of string suffixes in lexicographical order. For instance, given the string "patata", its suffixes in lexicographical order are *["a" (index 5), "ata" (index 3), "atata" (index 1), "patata" (index 0), "ta" (index 4), "tata" (index 2)]*, so its suffix array is *(5, 3, 1, 0, 4, 2)*. As identically duplicate samples have the same prefix, they will become adjacent in the suffix array, making it easier to find the duplicates across the samples. In practice, they construct a suffix array on the sequence with a threshold of 50 tokens (empirically determined for significantly reducing the false positives), and find the duplicate samples with common prefixes in linear time.

Approximate Hashing-based Deduplication. Hashing-based methods hash each sample into a fixed-length vector and deduplicate samples with significant vector overlap. Compared with the exact matching-based approach, it can identify near-duplicate samples with only a few words of difference (e.g., advertisements generated using the same template). Unlike normal hashing algorithms like MD5, hashes generated in this approach do not change significantly with even a bit of modification, making it possible to detect near-duplicate samples. There are various hashing algorithms, including SimHash [88], MinHash [81], DotHash [298], and their variants [347], [358].

• *MinHash [81]* hashes samples into vectors using a series of hashing functions, where only the minimum value is retained for each function, and estimates similarity for each pair of vectors through Jaccard Index $Jaccard(X, Y) = \frac{X \cap Y}{X \cup Y}$, where X and Y represent sets of elements (For example, if X = a, b, c, d and Y = b, c, d, e, f, the Jaccard Index over X and Y would be $\frac{1}{2}$). [356] demonstrates that MinHash generally outperforms SimHash. In practice, [122] employed MinHash to the code data on both the sample and the repository levels for diversity and integrity, and [299] employed MinHash on the sample level.

Moreover, MinHash has various variants for acceleration. MinHashLSH [347], [358] improves MinHash by involving locality-sensitive hashing (LSH), which divides a vector into multiple bands and only compares the samples with partially identical vector bands instead of the whole vector, mitigating the computational overhead in sample comparison. LSHBloom [207] further improves MinHashLSH by using Bloom Filter, which hashes each band into a single integer value and inserting it into each corresponding Bloom Filter, and the sample will be flagged as a duplicate if any band's hashed value collides with an entry in the Bloom filter, accelerating duplicate samples searching while reducing memory usage with negligible false positive rate (e.g., 1e-5 in experiments).

However, MinHash-based methods require building massive vector sets. When the number of samples and their lengths grow large, constructing vector sets becomes exceedingly expensive in terms of both time and space. Moreover, as the feature vector computation for each sample depends on this shared vocabulary, it is difficult to fully parallelize the process.

• *SimHash [88].* To address MinHash's issues, SimHash [88] generates a sample's feature vector *solely from the words it contains*, converts each sample into a fixed-dimensional binary vector for similarity comparison. Specifically, it first hashes each token in the sample (e.g., by BPE tokenizer [75]) into a fixed-dimension vector of $\{0, 1\}^d$ (e.g., $[1, 0, 0, 1]$ and $[1, 1, 0, 0]$) weighted by the pre-defined weight w (e.g., w_1 and w_2), where the weight is positive for 1 and negative for 0 (e.g., $[w_1, -w_1, -w_1, w_1]$, $[w_2, w_2, -w_2, -w_2]$). Then it added up these weighted vectors to a new vector of the same dimension d (e.g., $[w_1 + w_2, -w_1 + w_2, -w_1 - w_2, w_1 - w_2]$). Finally, the values of the new vector are mapped to another vector of $\{0, 1\}^d$, where the positive values are mapped to 1 and 0 otherwise. The final vector is the fingerprint of each sample, and the similarity of the two samples is estimated by calculating the Hamming distance between their vectors.

Compared with MinHash, SimHash stores and compares only one hash signature for each sample, greatly reducing the storage and computing overhead. However, keeping only one signature makes it harder to distinguish between two samples, especially those with low Hamming distances, requiring careful curation of data features.

• *DotHash [298].* Moreover, to further improve the deduplication accuracy and efficiency, DotHash [298] assumes that uniformly sampled vectors in high-dimensional space are quasi-orthogonal. It encodes each sample into a combination of sample elements represented as fixed-length basis vectors, and the dot product of these vectors is an unbiased estimate of their intersection. For example, given two samples with their element basis vectors $a = \sum_{a \in A} \psi(a)$ and $b = \sum_{b \in B} \psi(b)$, the intersection is calculated by $\mathbb{E}[a \cdot b] = |A \cap B|$.

However, [121] found that DotHash performs badly if the length of the basis vector is lower than the number of basis vectors, where quasi-orthogonal no longer holds.

Approximate Frequency-based Down-Weighting. To prevent the loss of potentially valuable information by retaining only one sample and removing the rest, SoftDeDup [167] deduplicates by reweighting samples, where samples with higher commonness are assigned lower sampling weights. Specifically, SoftDeDup computes the frequency of each n-gram across all the samples and calculates the commonness

of each sample by multiplying the frequencies of all the n-grams that appear in the document. Samples with higher commonness are more likely to be duplicates and thus be down-weighted.

Embedding-Based Clustering. Except for samples with the same or similar substrings, some samples with similar semantics but different formats (i.e, expressed differently) may also negatively affect LLM training performance. For instance, for the following two sentences: (*i*) *"Unleash your potential with our lightweight, high-performance sports shoes – designed for comfort, speed, and style"*; (*ii*) *"Step into greatness with durable, breathable sports shoes perfect for running, training, and everyday adventures"*. Both of the sentences are sports shoe advertisements but expressed differently, and such duplicates could degenerate model performance by making data imbalanced and introducing bias to the model. To address this issue, another approach leverages language models' embeddings (representing similar items as vectors close to each other in the vector space) for deduplication.

SemDeDup [46] identifies semantic duplicates by clustering embeddings and deduplicating those with high cosine similarities. It first encodes each sample into an embedding by leveraging the OPT [462] text encoder and the CLIP [325], [182] image encoder, and clusters the embeddings with K-means, so one can save time by finding duplicates within the cluster rather than the whole vector space. Then, within each cluster, it searches for semantic duplicates with cosine similarity above the pre-defined threshold. Finally, within each group of duplicates, it retains only the sample closest to the cluster centroid. As a multi-modal method, it can be applied to both text and image data, making it possible to deduplicate image data. In practice, [45] leverages SemDeDup to deduplicate the image-text pair dataset LAION-400M [341].

Like MinHash, SemDeDup also has many variants for performance improvement. [385] combines SemDeDup with the Self-Supervised Learning (SSL) Prototypes metric, which clusters the samples and retains the samples in each cluster based on their distance to their corresponding cluster centroid, where the samples closer to the centroid are more likely to be removed. FairDeDup [360] modifies the logic of SemDeDup to improve the representation of underrepresented sensitive groups by prioritizing the retention of samples that align with sensitive concepts defined through user-provided prototypes, such as demographic subgroups. Within each cluster, instead of selecting the farthest sample from the centroid, it selects the sample that maximizes similarity to the least-represented group in the cluster to prevent samples with sensitive concepts from being pruned.

Non-Text Data Deduplication. As LLMs are increasingly applied to multimodal tasks (e.g., image-text retrieval, visual question answering), non-text data types such as images are becoming integral to LLM training datasets, necessitating dedicated deduplication techniques. Similar to texts, images can also be encoded into embeddings through neural networks designed for image-like data such as CNN, after which embedding-based deduplication methods can be applied. *SemDedup* [46] adopts a semantic-based method by computing cosine similarity between image embeddings; two images are considered duplicates if their similarity exceeds a predefined threshold, which is tuned to balance detection precision and recall. In contrast, *MINT-1T* employs a hash-based approach, using SHA256 checksums to identify and remove exact duplicates efficiently. Meanwhile, the *DataComp* pipeline [146] leverages the CNN-based near-duplicate detector [445] to eliminate subtle duplicates and prevent evaluation set leakage. Models trained on these deduplicated image sets exhibit improved performance over baselines such as CLIP [325] for higher precision and recall.

TABLE 4: Data Filtering Methods for LLMs.

Category	Objective	Methods
Sample-level Filtering	Remove low-quality samples	Perplexity Measuring [383], [61], [288], [239], [238]
		Influence Assessment [254], [168]
		Clustering [45], [436]
		Model Scoring [411], [264], [345]
		Mixed Methods [285], [84], [126]
Content-level Filtering	Remove partial-noising samples	Privacy Anonymization [275], [268]
		Image & Video Filtering [437], [216], [390]

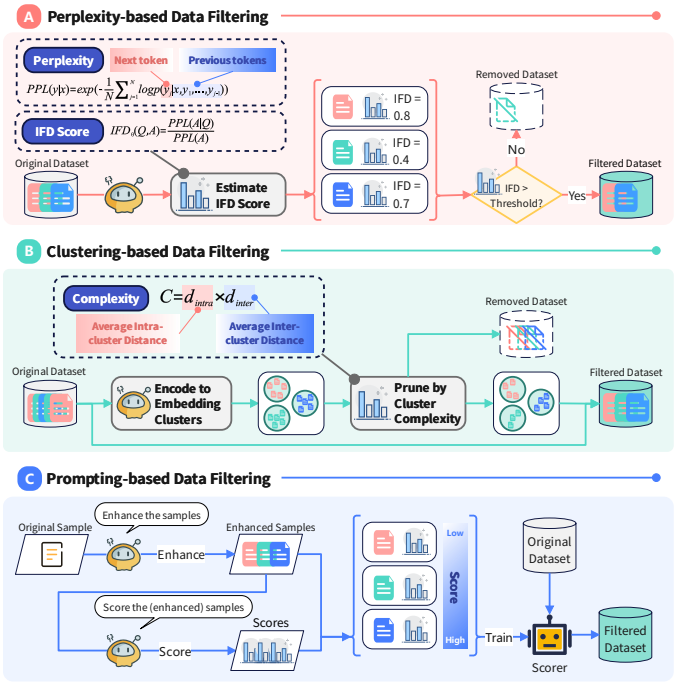

Fig. 5: Example Data Filtering Workflows [238], [45], [264].

2.3.3 Data Filtering

Data filtering removes low-quality or sensitive samples from the dataset to reduce computational overhead and protect privacy, while the model trained on the subset exhibits similar or even better performance than the one trained on the original dataset. To achieve this, one has to (*i*) remove samples with low quality (*Sample-level filtering*) or partial noisy information (*Content-level filtering*), and (*ii*) keep the selected samples diverse enough to cover various domains.

Sample-level Filtering refers to evaluating samples using metrics or models and removing the samples that fail to meet the threshold (e.g., quality and diversity). There are multiple metrics in this category:

Principles

Compared to classic ML data filtering, LLM data filtering emphasizes turning unstructured text into measurable metrics, with the main challenge being the effectiveness of evaluation methods, the standards of low-quality samples, and the computational complexity of these methods across massive datasets.

(1) Statistical Evaluation uses various statistical methods to evaluate samples by directly applying statistical metrics to the samples (e.g., clustering results) or indirectly capturing characteristics from the models trained on the dataset (e.g., loss or perplexity from a surrogate model). Applicable statistical metrics include perplexity (and its variants), influence on model parameters, and clustering.

• Perplexity Measuring. Perplexity measures the difficulty of a model generating the responses, represented as aggregated probabilities of the j-th response token given the question tokens and previous $j-1$ response tokens $\text{PPL}(y|x) = \exp\left(-\frac{1}{N}\sum_{j=1}^{N}\log p(y_j|x, y_1, ..., y_{j-1})\right)$. The higher the perplexity value is, the harder the model generates the response. It is commonly used in selecting high-quality subsets in pre-training and fine-tuning phases. Based on the original perplexity, there have been several studies for improving the metric, including computing perplexities using a smaller-sized model for training a larger-sized model to reduce computational overhead, or employing advanced techniques such as Learning Percentage (LP) and Instruction-Following Difficulty (IFD) to identify and select challenging samples.

Specifically, [383] uses an existing model to compute perplexity scores for multiple domains and selects pre-training samples from the domains with high correlation between the downstream benchmark error and the perplexity scores on the domain samples. The correlation is measured through a rank-based correlation coefficient $\gamma_j = \sum \text{sign}(y_k - y_l)(\text{rank}_j(x_{k,j}) - \text{rank}_j(x_{l,j}))$, where the rank difference reflects the model performance difference on the same sample, helpful in estimating θ^*. They then rank the domains based on γ_j and select samples from the top-ranked domains. To scale the process, a fastText classifier [199] is trained to distinguish selected documents, enabling page-level data selection.

To enhance efficiency, [61] leverages a smaller-sized surrogate model to select high-quality pre-training subsets via perplexity score for training larger-sized models, greatly reducing the computational overhead in model training while still achieving the same performance as with the full dataset. They first train a surrogate model, a smaller-sized MosaicML [378] model with 125 million parameters, on a random subset of the pre-training dataset to compute the perplexity scores for the remaining samples. Based on the perplexity scores, they find the optimal subset through a combination of selection criteria: (i) the part of samples to keep (e.g., samples with low/medium/high perplexity scores), and (ii) the fraction of samples to keep (e.g., 25%, 50%, 75%). The subset is evaluated by training a larger-sized MosaicML model on it and analyzing the model's performance on downstream benchmarks. While the result shows that the smaller-sized model can effectively and efficiently filter data for the larger-sized model, they also find that the effectiveness highly depends on the dataset. For example, keeping the high perplexity samples exhibits better performance on the Pile dataset [149], while keeping the medium perplexity samples exhibits better performance on the Dolma dataset [361].

Furthermore, there are some variants of perplexity-based evaluation. First, [288] proposes a perplexity-based metric, Learning Percentage (LP), to select samples that are more challenging for models to learn. Learning Percentage $\mathcal{LP}(i) = \frac{\mathcal{P}_{i-1}-\mathcal{P}_i}{\mathcal{P}_0-\mathcal{P}_n}$ measures the perplexity drop ratio of a sample between the specific epoch i and the whole training procedure. The key idea is that models tend to learn easier samples first and harder samples later, so one can find harder samples that are not thoroughly learned during early epochs. The authors use $\mathcal{LP}(1)$ (the learning percentage after the first epoch) to rank the training samples from the hardest to the easiest and split them into three equal-sized parts. It shows that the smaller-sized variant of the model can effectively select samples for the larger-sized variant, and models of all sizes trained on the harder part outperform the ones trained on all the samples.

Also based on perplexity, [239] proposes the Instruction-Following Difficulty (IFD) metric to select samples that are more difficult for models to follow. IFD ($\text{IFD}_\theta(Q, A) = \frac{PPL(A|Q)}{PPL(A)}$) measures the influence of the questions (instructions and inputs combined) on generating corresponding responses by comparing the perplexity of the response with or without the question strings $PPL(A|Q)$ and $PPL(A)$. A higher IFD score suggests higher model following difficulty. The authors first build a pre-experienced subset by clustering and resampling the samples from the WizardLM [426] and Alpaca-GPT4 [312] datasets, on which they train the model for one epoch to obtain initial knowledge. The model is then used to calculate the IFD score on all the samples, and the ones with high IFD scores are prioritized.

Superfiltering [238] further enhances [239] by employing the surrogate model from [61]. Instead of training a smaller-sized model, the authors directly use GPT-2 [327] as the surrogate model to calculate IFD scores on the same datasets. Compared to their previous work [239], the adoption of surrogate model simplifies the procedure and accelerates the filtering process.

• Influence Assessment. Another data filtering approach is to assess the influence of a sample on LLM model performance or learning process by measuring how the metrics change *when the sample is upweighted or removed*. The samples with substantial impact on the model parameters are regarded as influential and thus are selected.

DEALRec [254] identifies influential and challenging fine-tuning samples through two metrics: (i) *Influence Score* for assessing the influence of a specific sample on the model performance. It starts by measuring the influence on parameter change, where a surrogate model is trained on the full dataset to estimate how the model parameters would change when certain sample is removed or upweighted, expressed by $\hat{\theta}_{-s} - \hat{\theta} \approx \frac{1}{n}H_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(s, \hat{\theta})$, where $H_{\hat{\theta}}$ is the Hessian matrix and $\nabla_\theta\mathcal{L}(s, \hat{\theta})$ is the loss gradient of sample s. The formula is then evolved to measure the influence on empirical risk change, expressed by $I_{\text{remove, loss}}(s, \mathcal{D}) = \frac{1}{n}\sum_i \frac{1}{n}\nabla_\theta\mathcal{L}(s_i, \hat{\theta})^{\mathrm{T}}H_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(s, \hat{\theta})$; (ii) *Effort Score* for as-

sessing the difficulty for the surrogate model to learn a specific sample for generalization to new samples, defined as $\delta_s = \|\nabla_\phi \mathcal{L}^{\text{LLM}}(s)\|_2$, where Φ is the model parameter. A higher effort score suggests greater difficulty. The final score combines the above two scores, written as $I_s =$ Influence Score $+ \lambda \cdot$ Effort Score.

Besides, SHED [168] utilizes the Shapley value [339], which estimates the contribution of a member to the group, to calculate the influence of a sample on the model performance and select representative samples with high influence. The method first clusters the samples and selects the ones closest to each cluster centroid as the representative samples to reduce computational overhead. It then calculates the Shapley value for each representative sample i by iteratively removing n samples from the dataset until all the samples have been removed and calculating the contribution of the removed n samples in each iteration a to the model performance compared with the previous iteration, written as: $c_{(an+1..(a+1)n)\in D_p} = v(D_p \setminus \{1..an\}) - v(D_p \setminus \{1..(a+1)n\})$. The process will be repeated for k times for higher accuracy, after which the Shapley value for each representative sample i is defined as $S_i \approx \frac{1}{k}\sum_k \frac{c_i(k)}{n}$. Finally, the subsets can be selected either by selecting from the top-rank samples or weighted sampling the samples through $\Pr(i) = \frac{e^{fS_i}}{\sum_i e^{fS_i}}$, where f controls the trade-off between quality and diversity.

• Clustering. A common approach to select high-quality and diverse subsets is to encode the samples into embeddings in the latest space and cluster them using cosine similarity, where similar samples are usually clustered into the same group. Selecting within the clusters reduces redundancy, while selecting across the clusters increases diversity.

Density-Based Pruning (DBP) [45] selects high-quality and diverse subsets by clustering samples into clusters and resampling the samples based on the cluster complexity. They encode the samples into embeddings using a pre-trained vision model DINOV2-L/14 [300] and cluster them using K-means. For each cluster, they calculate the average intra-cluster cosine-distance to the internal centroid d_{intra} and inter-cluster cosine distance to the other centroids d_{inter}, and the cluster complexity as a product of the two distances $C = d_{intra} \times d_{inter}$. The cluster complexity is later converted to probability using softmax to resample the samples across clusters, where clusters with higher complexity have higher weights.

Rather than the sample embedding itself, SmallTo-Large [436] selects a diverse subset by clustering the samples based on their loss trajectories. It first trains a smaller-sized surrogate LLM model on the whole dataset to obtain the loss trajectories of each training sample, defined as $\mathcal{L}_i(\boldsymbol{\phi}^{(t)}) = -\log p_{\boldsymbol{\phi}^{(t)}}(\mathbf{y}_i|\mathbf{x}_i)$, where $\boldsymbol{\phi}^{(t)}$ is the model parameters at time t. These samples are then clustered based on loss trajectories and randomly resampled to form a diverse subset.

(2) Model Scoring uses LLMs for evaluating sample quality. The quality criteria can either be specified (i) explicitly via LLM prompt engineering or (ii) implicitly learned from human-labeled data.

QuRating [411] selects high-quality pre-training samples by prompting LLM to compare pairs of samples along the four quality criteria (writing style, fact & trivia amount, educational value, and the expertise required to understand), training a rater on the scalar quality ratings, and filtering samples using the rater. Initially, GPT-3.5-turbo is prompted on each pair of samples to judge which one is better on each quality criterion, where the binary confidence $p_{B\succ A} \in [0,1]$ that the sample B is preferred over the sample A is recorded. The pairwise binary confidence is then translated into sample quality ratings $p_{B\succ A} = \sigma(s_B - s_A)$ through the Bradley-Terry model. A QuRater model is later trained on these quality ratings to predict quality ratings for new samples on each criterion. The new samples are resampled with the probability $p(d_i) \propto \exp\left(\frac{s_i}{\tau}\right)$, where τ adjusts the trade-off between quality and diversity.

Rather than prompting the models to compare samples, Data-Efficient Instruction Tuning for Alignment (DEITA) [264] prompts LLM models to evolve and score the samples for building sample scorers. The authors first prompt ChatGPT to evolve the samples along instruction complexity and response quality, and again prompt ChatGPT to score these evolved samples. They then train scorers on the evolved samples with their corresponding scores to enable their scoring abilities. Finally, they use these scorers to score new samples and multiply the scores to form the final score, where the new samples are resampled based on the final scores for diversity.

Model scoring methods also help mitigate bias and toxicity. LLM often exhibit harmful biases due to the massive and unchecked datasets they are trained on, which can have various biases, ranging from gender and racial stereotypes to cultural and socioeconomic prejudices [296]. Safety-enhanced Aligned LLM Fine-tuning (SEAL) [345] selects high-quality and safe fine-tuning samples through a safety-aligned selector. The selector is trained based on a safety-aligned model, Merlinite-7b [366], using bi-level optimization, which minimizes the safety loss on the safe dataset while minimizing the fine-tuning loss on the filtered dataset during training to ensure the selector always prioritizes safe and high-quality samples during selection. After the selection, the top-p% samples will be selected.

(3) Hybrid Methods. Instead of relying on a single method, some methods mix various kinds of data filtering methods and evaluate each permutation of these methods or parameters to find the best combination of methods or parameters that further boosts model performance.

[285] selects high-quality pre-training data based on three metrics: (i) Perplexity, (ii) EL2N $\chi(x_i, y_i) = \mathbb{E}\|f(x_i) - y_i\|_2$ for measuring the prediction probability discrepancy between the reference model and the ground truth, and (iii) Memorization factor $score(M,N) = \frac{1}{N}\sum_i^N 1(z_{M+i} = \hat{z}_{M+i})$ for measuring the fraction of N tokens correctly generated after prompting the model with the first M tokens [77]. For each metric, they retain samples based on two criteria: (i) the fraction of samples to keep (10%, 30%, 50%, and 70%) and (ii) the part of samples to keep, e.g., the bottom (for Perplexity and L2-Norm Error) and top (for Memorization). They train LLM for each case and select the best-performing one, and the result shows that *Perplexity* effectively removes the "easiest" samples, improving model performance and outperforming other metrics.

Instead of comparing metrics and choosing the best of them, InstructionMining [84] combines various metrics (e.g., including input/output length, reward score, perplexity, etc.) into one linear function with each metric as indicator, written

as $logL_{loss} \propto L_0 + \beta_0 + \beta_1 I_1 + \beta_2 I_2 + \cdots + \beta_n I_n + \epsilon$. The β parameters are estimated using least squares. In practice, it evaluates fine-tuning samples on a fine-tuned model LLaMA-2-7B [386] and selects samples by finding the optimal set of samples to keep using the hyperparameter optimizer BlendSearch [395].

MoDS [126] considers diversity into selection and iteratively selects high-quality, diverse, and necessary subsets and adds the samples the LLM model performs poorly on during fine-tuning using a reward model and the K-Center greedy algorithm [342]. The method is conducted mainly in three steps: (*i*) Use a reward model to score the quality of each (instruction, input, output) triplet in the dataset, where the low-quality ones are filtered out, forming a high-quality dataset. (*ii*) Use the K-Center greedy algorithm [342] to select the samples in the high-quality dataset that are farthest apart from each other in the BERT [206] embedding space, forming a diverse seed dataset. (*iii*) Fine-tune a pre-trained LLM model on the seed dataset to enable its instruction-following ability and generate responses for the high-quality dataset. The generated responses are evaluated using the same reward model, and those with low quality scores, which means the model is weak at generating such responses, will be collected. The collected samples with their original responses will be selected again using the K-Center greedy algorithm and then added to the seed dataset, forming the final dataset.

Content-level Filtering. To avoid removing too many critical samples from the dataset and weakening the model performance, some works only filter out noise or sensitive content within the samples. For noise removal, common methodologies include removing or replacing specific characters (e.g., remove invisible or invalid characters, unescape HTML characters and detect punctuation misuse), removing unnecessary texts (e.g., the texts that appear as decorating elements on the web pages such as "print", "likes" and "loading"), and cleaning harmful information (e.g., spam, gambling, pornographic content and site links) [433].

For privacy anonymization, LLMs can memorize private and sensitive information (e.g, user identity details or clinical health data) from datasets during pre-training and fine-tuning, which can be leaked through specially crafted prompts, thereby posing significant privacy risks. [275] demonstrates that it is possible to extract, reconstruct, and infer personally identifiable information (PII) from LLM models by identifying the most frequent PII appearing in model responses or by prompting models with partial information about a specific individual. From a data management perspective, these privacy threats can be mitigated by identifying and filtering out potential sensitive information in the datasets.

DeID-GPT [268] utilizes existing LLMs to identify and remove PII from unstructured medical text without changing its meaning. In their case, the LLMs are prompted to de-identify information from clinical notes in accordance with HIPAA privacy regulations. An example prompt is: *"Please de-identify the following clinical notes by replacing any terms that could be a name, an address, a date, or an ID with the term '[redacted]':"*

Instead of using general LLMs, [275] uses Named Entity Recognition (NER) models such as spaCy [33] and Flair [52] to tag PII in the samples and removes or replaces them with hashed tags, entity tags like "[NAME]" or "[LOCATION]", or a simple tag like "[MASK]". The last tag was adopted to maximize privacy, as the other ones are still vulnerable to membership inference by linking the samples.

TABLE 5: Comparison of Different Data Selection Methods.

Method	Stage	Evaluation Metric
Similarity	Pre-training, Fine-tuning	Cosine Similarity [423]
		Bag-of-Words Similarity [421]
		Lexicon Set Overlap [321]
		Bayes-based Selection [80]
Optimization	Fine-tuning	Linear Search [130]
		Gradient-Influence Search [417]
		Kernel-Density Regularization [269]
Model	Pre-training	Logits-based LM-Score [465]

The rise of multi-modal LLMs, particularly large video generation models, drives the need for robust video data filtering. CogVideoX [437] employs a pipeline focusing on coherent motion, removing videos with poor dynamics. It defines negative labels for artificial edits, low motion connectivity, visual flaws, and excessive text. A manually annotated subset trains six Video-LLaMA[455]-based filters, while optical flow and aesthetic scores ensure motion coherence and visual appeal, refining the dataset to approximately 35M high-quality 6-second clips.

HunyuanVideo [216] uses a multi-step pipeline: splitting videos into clips, encoding embeddings, deduplication, and resampling. Filters include motion (OpenCV-based optical flow), OCR (text removal), clarity (visual blur detection), aesthetic (Dover[414]-based scoring), and source (YOLOX[153]-like watermark/border removal). This process generates five progressive training sets with increasing thresholds.

Wan [390] applies pre- and post-processing pipelines. Preprocessing filters unsuitable data using OCR, aesthetic evaluation (LAION-5B [341]), NSFW scoring, watermark detection, and resolution thresholds, removing approximately 50% of low-quality data. Samples are clustered for diversity, manually scored, and an expert model selects high-quality, naturally distributed data. Videos are classified into six tiers, prioritizing smooth motion. Post-processing refines images by selecting top 20% via an expert model and manually curating gaps. For videos, top candidates are filtered by visual quality and motion complexity, ensuring balance and diversity across 12 themes.

2.3.4 Data Selection

Different from previous reviews [55], [398], we define data selection as the process of choosing subsets of already well-cleaned data samples in order to adapt LLMs to specific domains (e.g., medical or legal LLMs).

Principles

Unlike traditional ML data selection, LLM data selection focuses on aligning the topics of the text samples, requiring encoding semantic topics into measurable distributions. However, managing computational efficiency and ensuring robust generalization across diverse tasks remain critical unresolved issues.

Similarity-based Data Selection. One class of methods aims to select subsets similar to the specified target data.

• *Cosine Similarity:* Domain-Adaptive Continual Pre-training (DACP) [423] adapts a general-purpose LLM to a target task by selecting domain-specific unlabeled data based on similarity (cosine similarity), novelty (perplexity), and diversity (entropy). For the similarity part, it identifies data most similar to the task-specific labeled data by encoding both into embeddings (using [33]) and choosing domain samples that align with the task's embedding distribution.

• *Bag-of-Words Similarity:* DSIR [421] selects a subset of unlabeled pre-training data matching the target distribution by computing feature distributions ($\hat{p}_{\text{feat}}$, $\hat{q}_{\text{feat}}$) for raw and target data represented as bag-of-words, estimating importance weights $w_i = \frac{\hat{p}_{\text{feat}}(z_i)}{\hat{q}_{\text{feat}}(z_i)}$, and resampling raw data with probability $\frac{w_i}{\sum_{i=1}^{N} w_i}$.

• *Lexicon Set Overlap:* [321] selects the subset with the most shared lexicons using the Domain Specific Score (DSS), which quantifies the relevance of a dialogue set T to specific domains by measuring the overlap between T and domain lexicons $L = \{l_1, l_2, \ldots, l_m\}$, calculated as $\text{DSS}(T, L) = \frac{1}{m}\sum_{i=1}^{m} \frac{|T \cap l_i|}{n}$, where n is the number of tokens in T.

• *Bayes-based Selection:* CoLoR-filter [80] formulates pre-training subset selection as a Bayesian optimization problem, which selects a subset S by maximizing downstream likelihood $\Pr(D_{\text{down}}|S)$. It uses two auxiliary models: A "prior" model (θ_{prior}) trained on a large general dataset D_{down} and a "conditional" model (θ_{prior}) fine-tuned on the union of the large general dataset and a small downstream dataset $D_{\text{prior+down}}$. The selection criterion for a data point x_i is the conditional loss reduction (CoLoR): $\text{CoLoR}(x_i) = -\log \Pr(x_i|\theta_{\text{prior+down}}) - (-\log \Pr(x_i|\theta_{\text{prior}}))$. The key idea is to score samples based on the likelihood difference between these two models and select the ones that exhibit higher likelihood under the conditional model and larger conditional loss reduction.

Optimization-based Data Selection. Optimization-based data selection methods select subsets towards reducing model loss and improving model performance on the target tasks.

• *Linear Search.* Model-Aware Dataset Selection with Datamodels (DsDm) [130] selects the optimal subset of training data that minimizes the model's loss on target tasks by employing linear datamodel [184], a parameterized function that maps a subset of training data to the model outputs for the specified target, to estimate how the inclusion of each training sample would affect the model's loss on the target, reducing computational overhead. In practice, a linear datamodel $\tau_{\theta_x}(1_S) = \theta_x^\top 1_S$ with parameters θ_x and a characteristic vector 1_S (a binary vector indicating which samples are in S) is adopted to map the subset S to the model loss on a sample x through $L_x(S) = \mathbb{E}[\ell(x; A(S))]$. For each target, the characteristic vector 1_S is adjusted to reflect the subset, and the parameters θ_x are estimated using a regression loss function like mean squared error over the training subset. After training, the datamodel selects the subset S of the size k that minimizes the loss $\hat{L}_{D_{\text{targ}}}(S) = \frac{1}{n}\sum_{i=1}^{n} \tau_{\theta_{x_i}}(1_S)$ for the target task.

• *Gradient-Influence Search.* Low-rank Gradient Similarity Search (LESS) [417] identifies the most impactful subset of data for fine-tuning LLMs by analyzing gradient similarities. It first fine-tunes the model on a random subset (e.g., 5% of data) for a few epochs using LoRA to reduce trainable parameters and accelerate gradient computation, and saves the checkpoints after each epoch. Next, LESS computes Adam LoRA gradients for each training sample, projects them into lower-dimensional gradient features via random projection, and stores them in a gradient datastore. For downstream tasks, it calculates gradient features of few-shot validation samples and estimates the influence of each training sample $\boldsymbol{z}$ on a validation sample $\boldsymbol{z}'$ using cosine similarity: $\text{Inf}_{\text{Adam}}(\boldsymbol{z}, \boldsymbol{z}') \triangleq \sum_{i=1}^{N} \bar{\eta}_i \cos(\nabla \ell(\boldsymbol{z}'; \boldsymbol{\theta}_i), \Gamma(\boldsymbol{z}, \boldsymbol{\theta}_i))$, where $\Gamma(\boldsymbol{z}, \boldsymbol{\theta})$ is the Adam update. The training samples with the highest influence scores are selected for fine-tuning.

• *Kernel-Density Regularization.* Task-Specific Data Selection (TSDS) [269] identifies high-quality pre-training or fine-tuning data for particular tasks by balancing two objectives: (i) distribution alignment with the target task data and (ii) diversity to avoid near-duplicates, accomplished via kernel density estimation (KDE) regularization. Concretely, one begins with a small set of target task samples $Q = \{q_i\}_{i=1}^{M}$ and a large candidate pool $D = \{x_j\}_{j=1}^{N}$, both of which are embedded into a shared metric space (e.g., using gradient-based or semantic embeddings). The optimization for distribution alignment is conducted by solving for probability mass γ_{ij} (transported from q_i to x_j): $\min_{\gamma \in \mathbb{R}_{\geq 0}^{M \times N}} \frac{\alpha}{C}\sum_{i=1}^{M}\sum_{j=1}^{N} \gamma_{ij} d_{ij} + (1-\alpha) G_{\text{KDE}}(\gamma)$ s.t. $\sum_{j=1}^{N} \gamma_{ij} = \frac{1}{M}, \forall i \in [M]$, where d_{ij} is the distance between q_i and x_j in the metric space, and $G_{\text{KDE}}(\gamma)$ is the regularization term that adds diversity and penalizes over-density using KDE: $G_{\text{KDE}}(\gamma) = M \max_{i,j} \rho_j \left| \gamma_{ij} - \frac{1/\rho_j}{M \sum_{j'} 1/\rho_{j'}} \right|$, where $\rho_j = \sum_{x' \in D} (1 - f(x_j, x')^2/h^2$ is the density estimate for candidate x_j (higher for near-duplicates). Afterwards, it samples x_j with probability $p_j = \sum_i \gamma_{ij}^*$.

Model-based Data Selection. These methods aim to determine subsets guided by prompting the LLM itself.

Autonomous Data Selection (AutoDS) [465] prompts the LLM to assess and select mathematical and educational samples from a larger dataset. For each sample, the LLM is asked two questions: (i) Is it mathematically relevant, and (ii) It it educationally valuable. The LLM responds to each question with "Yes" or "No", and the logit of each response is extracted to compute the LM-Score: $\text{LM-Score}(\cdot) = \frac{\exp(\text{logit}(\text{'YES'}))}{\exp(\text{logit}(\text{'YES'})) + \exp(\text{logit}(\text{'NO'}))}$, and the composite score: $\text{LM-Score}(Q_1, Q_2) = \text{LM-Score}(Q_1) \cdot \text{LM-Score}(Q_2)$. The composite score ranks and selects high-quality math samples.

2.3.5 Data Mixing

For the **IaaS** data concept, this section mainly focuses on the Inclusiveness of data, specifically, the construction of a purposefully balanced composition.

Since LLMs rely on massive and diverse datasets, the composition of these datasets significantly impacts model performance [295]. For instance, as shown in Figure 3, we can see LLMs require different ratios of domain data to achieve capabilities such as medical diagnosis, coding, and solving math problems. To this end, data mixing refers to the strategy of (1) combining datasets from different domains, sources or structures in specific proportions to train LLMs or (2) making

TABLE 6: Comparison of Data Mixing Methods for LLMs.

Taxonomy	Stage	Methods	Traits
Before Training (Human Experience)	Pre-training	Multi-Source Data Adjusting [139], [347]	Intuitive and easy to implement, suitable for rapid experimentation.
		Entropy-Based Mixing [152]	Low computation cost with quality quantification by entropy.
Before Training (Model-Based Optimization)	Pre-training	Linear Regression Model [263]	Only 10% of DoReMi's [420] computational resources are required. Simultaneously train hundreds of small models to accelerate optimization.
	Pre-training	Bivariate Data Mixing Law [152]	Avoid iterative training of proxy models (low computational costs). Show relation between loss and training steps.
	Continual Pre-training	Chinchilla Scaling Law [323]	Support knowledge transferring to new domains (↓ over 95% training costs).
	Pre-training	Exponential Functions [439]	Support datasets without explicit domain division.
	Continual Pre-training	Power-law Function [160]	Compared to single-objective optimization like [323] [160] ensures that domain performance improvement does not compromise general capabilities.
	Pre-training	Classification Model [251]	Reverse engineering for finding the suitable data recipe of LLMs.
During Training (Bilevel Optimization)	Pre-training	Calculate domain contribution by gradient inner products[135]	Requires a proxy model, performances well in OOD datasets.
	Fine-tuning	Dynamically adjust weights by gradient alignment values [302]	Multiple applications like multilingual training, instruction following, large-scale data reweighting
During Training (Distributionally Robust Optimization)	Pre-training	Group DRO [420]	For pre-training, smooth adjusting to prevent abrupt weight changes
	Fine-tuning	Task-level DRO [278]	For fine tuning, quick response to task difficulty changes

LLMs give different proportions of attention on different domains (e.g., by changing the sampling probabilities) in the training session. Effective data mixing ensures that the model captures broad generalization capabilities while balancing performance across tasks and domains [140]. Existing data mixing methods can be classified into two main categories:

Principles

Unlike traditional ML models like BERT (trained on smaller, domain-specific data with homogeneous distributions), LLMs require massive multilingual or multi-domain corpora, raising the critical challenge of optimizing dataset mixing ratios for performance. Current methods use heuristic experimentation or formulate ratio-performance relationships (e.g., validation loss), but cost-effective determination of optimal ratios, beyond heuristics, remains unresolved due to high cost demands for functional approximations.

Before-Training Mixing (Human Experience). This method provides empirical data mixing strategies such as setting different ratios of datasets based on various factors (e.g., complexity and diversity of the datasets) that likely improve LLMs' abilities.

First, to study the effect of data mixture, there are works that experiment heuristically on different data ratios for pre-training of LLMs. [139] suspects training sequence from simple to complex data would improve LLMs' performance, thus introduces a two-stage data mixing strategy for LLM pre-training: (1) It first blends web-crawled data with minimal high-quality content (1.9% math, 15% code), testing ratios (<35% high-quality) and selecting optimal mixtures via evaluations on CommonsenseQA [371] and HumanEval [95]. (2) It then filters low-quality data, boosting math (24%→29%), code (20%→29%), and instructional alignment data. Ratios are similarly optimized through empirical validation. The method iteratively refines proportions using down-sampled Megatron-8B [355] for efficiency, then scales findings to a 25B model, balancing diversity-quality tradeoffs with reduced experimental overhead. Similarly, Slimpajama [347] explores the impact of data source diversity and weight distribution on model performance by adjusting the proportions of data from multiple sources, such as Commoncrawl [11], C4 [330], Github [14] .

Second, we can utilize metrics to judge different datasets and mix them. To calculate the best result rather than just try different combinations, Bimix [152] adopts entropy metrics (e.g., Shannon entropy [343], conditional entropy [343]) as the quality scores which are then normalized to compute the proportions of each domain (e.g., conditional entropy, written as as $H_i\left(X_i^{(t+1)} \mid X_i^{(t)}\right) = -\sum_{x \in X_i^{(t)}} \sum_{x' \in X_i^{(t+1)}} P(x, x') \log P(x' \mid x)$, where $X_i^{(t+1)}$ $X_i^{(t)}$ are sets of tokens at positions $t+1$ and t separately, x and x' are tokens belonging to them, $P(x, x')$ is the joint probability, $P(x' \mid x)$ is the conditional probability.

Before-Training Mixing (Model-Based Optimization). This category of methods design linear or non-linear models that depict (i) the relation between the distribution of each domain, (ii) validation loss, and (iii) some other variables like training steps, based on which they find the optimal settings through various model-based techniques.

(1) Linear Regression Model: Some methods utilize pairs like data mixtures and corresponding model performance to fit a linear regressing model, such that finding the best data mixture ratios.

Typically, REGMIX [263] defines the domains by source (like ArXiv, FreeLaw, etc.), which uses Dirichlet distribution (which controls the distribution of probabilities across multiple categories with a parameter) to generate all kinds of data distribution of several domains to train a small-scale proxy model to collect performance data, which is then used to fit a linear regression model (LightGBM [205]) to predict the optimal data mixing distribution. Then REGMIX uses both the best distribution and the average of top-100 distributions to verify on variations of TinyLlama [459] with additional layers with versions of 1B and 7B.

(2) Non-linear Regression Model: There are also many methods that design non-linear regression models for data mixing by considering more complex training characters.

• *Bivariate Data Mixing Law.* Based on observations of validation loss changes due to variables like domain proportion (where the data come from different sources like Pile-CC) and training steps, BiMix [152] proposes Bivariate Data Mixing Law that depicts the relation among domain's proportion, training steps and validation loss, which can be written as $L_i\left(r_i, s\right) = \frac{A_i}{r_i^{\alpha_i}}\left(\frac{B_i}{s^{\beta_i}} + C_i\right)$, where A_i,B_i,C_i are domain-dependent scaling coefficients, α_i and β_i are power-law exponents that control the influence of domain proportion and training steps respectively, s represents the training step count. It utilizes the law to fit the actual data curves by fixing the domain's proportion or training steps and varies the other one to get validation loss by training a small model (decoder-only transformers based on the DoReMi [420] architecture with 280M parameters). After depicting the relation, we model the task as an optimization problem (resolvable by Lagrange multipliers) and then verify on larger LLM (decoder-only transformers based on the DoReMi [420] architecture with 1B parameters).

• *Chinchilla Scaling Law.* D-CPT [323] establishes a mathematical relationship which could be used to find the best mixture of general and domain-specific data between validation loss, model size, data size, and domain data mixing ratios based on Chinchilla Scaling Law [170] to optimize domain-specific continual pre-training as $L(N, D, r) = E + \frac{A}{N^\alpha} + \frac{B \cdot r^\eta}{D^\beta} + \frac{C}{(r+\epsilon)^\gamma}$ (N is model parameter count, D is training data volume (number of tokens), r is domain corpus ratio, $E, A, B, C, \alpha, \beta, \gamma, \eta, \epsilon$ are fitting parameters), with a variation which introduces K which describes the difficulty to learn the domain's knowledge as $L(N, D, r) = E + \frac{A}{N^\alpha} + \frac{B \cdot r^\eta}{D^\beta} + \frac{C}{(r+\epsilon)^\gamma} + \frac{F}{K^\mu}$ (F is a fitting parameter). It fits formula parameters through small-scale experiments to predict performance under different training configurations and find the suitable ratio to minimize the domain validation loss while ensuring the generalization loss does not exceed the specified threshold.

• *Exponential Functions.* Data Mixing Law [439] establishes an exponential relationship between validation loss and data mixing ratios of several domains (e.g., public datasets like Pile-CC, Books3), $L(r) = c + k\exp\left(\sum_i t_i r_i\right)$, where $L(r)$ is the validation loss, r represents the mixing ratios of different domains, and c, k, and t_i are learnable parameters. That is, it experiments on a small model with the exponential relationships to predict the best data domain mixing ratios on LLM performance with scaling laws, which combines training step scaling laws ($L(S) = c + kS^\alpha$, where S is the number of training steps, and α is a fitting parameter.), which is used to infer the validation loss at target training steps from results at smaller steps, and model size scaling laws ($L(N) = c + kN^\beta$, where N is the number of model parameters, and β is a fitting parameter), which is used to infer the validation loss for large model sizes from smaller model sizes.

• *Classification Model.* [251] aims to find the data proportion of closed-source model by data proportion detection, which first generating large-scale data from the LLM, then using a classification model to categorize the generated data and compute perplexity, deriving the proportions of pre-training data based on the Data Mixing Law [439] (which is a mathematical formula describing the relationship between the proportion of pre-training data and the model's loss in different domains.).

• *Power-law Function.* CMR [160] aims to optimize the continual pre-training by finding the best ratio of generic dataset and domain-specific dataset. Based on the research before and the data observed on different sizes of models with different ratios of data, the relationships between loss and mixture ratio, and training volume fit in power-law forms, which are described as $L(R) = \alpha \cdot R^s + \beta$ and $L(T) = \alpha_1 \cdot T^{s_1} + \beta_1$, where α, β, s, α_1, β_1 and s_1 are fitting parameters. Based the relationships, they propose a metric *Critical Mixture Ratio*, which is the maximum data mixing ratio that balances between (1) significantly reducing domain loss while (2) keeping the increase in general loss within a pre-defined tolerance range. Based on the two aspects, the ratio is defined as $R^* = \max\{R \mid R \in F\}$, where R is the ratio of generic dataset and domain-specific dataset, F is feasible mixture ratios which comprises all mixing proportions that satisfy the constraints of the general loss function.

During-Training Mixing (Bilevel Optimization). This method adopts a closed-loop optimization technique that ensures model parameters are well optimized [108]. Generally, Bilevel optimization involves two nested optimization problems: (1) the inner-level problem ensures model parameters are optimized under given weights (e.g., minimizing weighted training loss), while (2) the outer-level updates weights through backpropagation of validation loss, forming a closed-loop optimization.

Typically, ScaleBiO [302] reconstructs the data sampling weight optimization problem into a bilevel optimization problem, where outer-level problem is adjusting data weights to minimize validation loss; and the inner-level problem is adjusting model parameters to minimize weighted training loss and it could be applied to tasks like multilingual training (mixture of languages) and instruction following (mixture of quality). ScaleBio first experiments on small models. Then it extends to larger models like LLaMA-3. ScaleBiO initialize the weights equably for all data sources. In each iteration, it randomly selects a subset of data sources to update their weights: for the selected data sources, it adjusts the weights by optimizing the gradient of the validation loss, prioritizing the increase of weights for data that contribute significantly to model performance, while decreasing the weights for data that have less impact on performance. After updating the weights, retrain the model parameters and repeat the process until convergence.

To enhance the efficiency of BiO-based data mixing, DoGE [135] defines (i) inner-level problem as that under the condition of fixed data mixing ratios, optimize the proxy model parameters to minimize the weighted sum of domain losses; and (ii) outer-level problem as *adjusting the data mixing ratios such that the model parameters obtained through inner-level problem optimization achieve optimal performance on the target loss.* The method is executed on a small-scale proxy by following steps: Initially, it sets the domain weights as a uniform distribution. In each iteration, it dynamically adjusts the weight of each domain based on the gradient alignment value (calculated as the inner product of the gradient of current data domain and the sum of gradients from all data domains), which measures the contribution of the data from the current domain to the gradient direction of all other domains' data. Using the updated weights, it resamples the data and updates the model parameters. Repeat the process

for multiple iterations until the weights stabilize, then apply to actual LLM pre-training.

During-Training Mixing (Distributionally Robust Optimization). To search for a robust data mixing strategy (which can be sub-optimal but with low uncertainty), some methods adopt Distributionally Robust Optimization (DRO) for data mixing. DRO achieves robustness against distributional uncertainty by optimizing for the worst-case scenario within a set of distributions (referred to as the uncertainty set or ambiguity set).

• For LLM pre-training, DoReMi [420] defines the worst case as domains where the proxy model underperforms compared to the reference model, which initially sets the domain weights as a uniform distribution and each domains contains several sample sets, and uses it to train Transformer decoder-only LM with 280M parameters and computes loss in each example set, which provides a reference point to measure the improvement potential (the loss difference) of the proxy model in each domain. Next, DoReMi trains a small-scale proxy model (also Transformer decoder-only LM with 280M parameters) by adjusting the domain data weights through DRO, which dynamically adjusts the domain weights and tilt the weights toward domains with larger losses (compared to the reference model). Finally, validate performance of weighted domain data on large models (Transformer decoder-only LM with 8B parameters).

• For LLM fine-tuning, tDRO [278] defines the worst case the same as DoReMi, which computes the relative loss for each domain with a proxy model (e.g. Qwen1.5-0.5B [69]); and they compare the training loss of domain data with the reference model (e.g., Qwen1.5-0.5B), and evaluate each domain's potential for model improvement, and update the domain weights accordingly, giving more attention to high-loss domains. Finally, the updated weights are normalized to form a new sampling distribution and repeat the process to get final data distribution.

2.3.6 Data Distillation and Synthesis

For the **IaaS** data concept, this section covers a lot of points in **IaaS**, mainly focuses on the Inclusiveness, abundance and articulation of data. (*i*) For Inclusiveness, works in this section involves various domains, task types, expression styles, and modalities. (*ii*) For abundance, specifically for large-scale data, almost all works in 2.3.6 help to build the large scale of data in different degrees. (*iii*) For articulation, works in this section most help to improve the instructive and step-by-step reasoning part of the articulation. (*iv*) For Sanitization, there is one work at the very end of this section that helps protecting privacy.

Synthetic data, which mimics real-world scenarios, is particularly valuable for resolving problems such as (*i*) data scarcity (e.g., augmenting data for a small dataset) [426], (*ii*) privacy concerns (e.g., replacing sensitive data with synthesis data) [419], (*iii*) the need for diverse and high-quality datasets (e.g., generating examples for underrepresented cases) [260], (*iv*) lack of reasoning data (e.g., for code, chain of thought), (*v*) human alignment (e.g., label better LLM's response by human beings or LLMs).

Principles

Traditional ML methods use rule-based templates, basic augmentation (lexical substitution, back-translation), or statistical models to create limited synthetic data, addressing data scarcity/class imbalance. While LLM-driven synthesis employs LLMs to produce diverse, high-quality data, tackling data scarcity, privacy concerns, and diverse training needs. Key paradigms include: (i) sample-driven generation, (ii) domain-aligned synthesis, and (iii) reasoning-centric formatting. Challenges involve ensuring rigorous reasoning chain synthesis and optimizing cost-quality balance in data production.

Despite the advantages, synthetic data can negatively impact LLM training, such as when characteristics like toxicity are inherited from the source model or even amplified [352]. Thus, it is vital to design data synthesis methods for LLMs [495]. As shown in Figure 4, we discuss methods dealing these problem through the diverse LLM stages, including pre-Training, SFT, Reinforcement Learning and RAG.

Knowledge Distillation. Due to LLMs' massive parameter scale and high resource demands which make practical deployment challenging, so we utilize knowledge distillation (such as designing paradigms to prompt LLM to generate high-quality data) to training a student LLM with less parameters to mimic the target model's generation ability.

• *Task-Specific Prompt Distillation.* To significantly reduce inference costs and latency while maintaining performance, [353] employs task-specific prompts: (1) Chain-of-Density (CoD): Iteratively adds entities to summarize for enhanced density. (2) Chain-of-Thought (CoT): Guides reasoning tasks (e.g., math) through stepwise logic. Using GSM8K [106] data and Llama-3.1-405B-Instruct, synthetic data is generated for fine-tuning smaller models (Llama-3.1-8B/70B-Instruct) paired with simplified prompts, balancing efficiency and task specialization.

• *Code Verification and Error Correction Distillation.* Existing knowledge distillation methods (e.g., Chain-of-Thought Fine-tuning) rely on synthetic data generated by LLMs, but such data often contains incorrect intermediate reasoning steps which can mislead small models during learning, hindering the improvement of their reasoning capabilities.

Pad [496] proposes Program-aided Distillation (PaD) to address error-prone synthetic data in knowledge distillation with (i) Programmatic Reasoning: LLMs generate executable code (e.g., math problems as Python calculations) instead of natural language CoT, with Python compilers auto-filtering logic errors. (ii) Error-Injection Training: Models learn error correction by fixing synthetically injected AST-based errors (e.g., NameError). (iii) Semantic Validation: Decoding selects steps via semantic alignment scoring (e.g., cosine similarity) to prevent error propagation. PaD replaces flawed CoT steps with verifiable program logic, enhancing small models' reasoning robustness through code-based distillation and self-correction mechanisms.

• *Multi-stage Collaboration Distillation Between Student models.* In domains with high annotation costs (e.g., biomedical parsing) or complex task structures (e.g., syntactic/se-

mantic parsing), labeled data is extremely scarce, making traditional supervised fine-tuning ineffective. MCKD [467] introduces Multi-stage Collaborative KD (MCKD) for low-resource generation as 3 steps. (i) Initialization: GPT-3.5 generates pseudo-labels for unlabeled data. (ii) Collaborative Distillation: Splits data into two subsets for cross-labeling via paired T5-Base models, reducing noise overfitting. Iteratively refines labels over 3 iterations. (iii) Final Training: Trains a single model on refined labels. Achieves near-supervised performance with 50 labeled examples (vs. 500 required traditionally) through multi-stage noise reduction and collaborative pseudo-label optimization.

Pre-training Data Augmentation. The pre-training stage of LLM requires a vast amount of data and it can be costly to synthesize such data with powerful models like GPT-4. Therefore, there are techniques like distillation [481], or simply mixing synthetic data into the whole corpus.

• *Distilled LLM for Mathematical Data Synthesis.* JiuZhang3.0 [481] proposes an LLM-based synthesis method for high-quality math problems: (i) Model Distillation, fine-tunes DeepSeekMath-7B on GPT-4-generated QA pairs (with curated prompts and math texts) to mimic GPT-4's generation. (ii) Uses gradient similarity to prioritize task-relevant data. (iii) Refines the model with filtered data to produce aligned outputs. The final math synthetic corpus are generated by the refined model based on the multi-source corpus (e.g., Wikipedia) and prompt sets.

• *Fintuned LLM for Instruction-Response Pair Synthesis.* In order to study the effect of supervised pre-training, Instruction PT [99] introduces an Instruction Synthesizer (Mistral-7B finetuned on 40+ task categories) to augment raw text with few-shot multi-task instructions (e.g., "Summarize school activities" → QA/reasoning pairs). Unlike GPT-style pre-training, it integrates structured task execution (QA, classification) alongside language modeling. This hybrid approach boosts data efficiency (500M model ≈ 1B baseline) and multi-task adaptability from pre-training.

• *LLM Prompting for Mathematical Data Synthesis.* Current math-specialized LLMs rely on SFT with problem-solving data (e.g., step-by-step solutions). However, since CPT improvements in math are far less significant than SFT gains.

To study the impact of problem-solving data in continual pre-training, [98] proposes enhancing models' mathematical reasoning capabilities by augmenting problem-solving data (e.g., step-by-step solutions for common math problems) during pre-training, rather than relying solely on traditional math corpora (e.g., theorem texts). First, a student model (Llama2 [386]) is utilized to generate answers from the collected math problems. Then, it uses a teacher model (Llama2 [386] with more parameters) detects errors in a student model's solutions and generates corrective steps guided by prompts. This teaches the target LLM self-checking and error-correction skills. Experiments indicate continual pre-training excels at learning complex reasoning (e.g., multi-step equation solving) than SFT, where MathGPT-8B using only 100B well-generated math-related tokens can exhibit capabilities comparable to Qwen2-Math-72B [434].

• *LLM Prompting for Rephrasing Synthesis.* To introduce more diversity to the data, some methods rephrase the data to different styles of texts like Q&A or concise definition. WRAP [282] leverages instruction-tuned models (e.g., Mistral-7B) to rephrase web text (C4) into four formats: (i) simple vocabulary and sentence structures that are understandable to young children. (ii) Standardized encyclopedia-style expression. (iii) Complex terminology and concise academic sentence structures. (iv) multi-turn dialogue. Mixing rephrased and original data trains LLMs to adapt to diverse formats (e.g., zero-shot QA), achieving 3× faster training and 50% lower perplexity on Pile benchmark [149] via hybrid real-synthetic data synergy.

• *LLM Prompting for Cross-language Synthesis.* LLMs like Llama-3 exhibit deficiencies in cross-language tasks and multidisciplinary scientific reasoning, while continual pre-training often triggers catastrophic forgetting (e.g., performance degradation in original capabilities like English tasks). [93] proposes to synthesize data so as to enhance Llama-3's Chinese proficiency and scientific reasoning capabilities while mitigating catastrophic forgetting. They utilize Mistral-7B [188] to generate multidisciplinary scientific question-answer pairs (e.g., Q&A on "explaining the electrostatic repulsion principle of ion double layers in electrolyte solutions") from seed data collected and classified into multiple disciplines by TinyBERT [195] and BERT-Tiny-Chinese [23] from Dolma's CC [361] and C4 [120]. And generate coding problems with LeetCode algorithm tasks as seeds by Magicoder-S-DS-6.7B [409] .These are mixed with Chinese, English, and synthetic data in a 1:7:2 ratio, significantly boosting scientific reasoning.

Additionally, through substitution experiments (validating data strategies using TinyLlama-1.1B [459] as a proxy model), they find that (1) a 20% synthetic data ratio with an error rate below 30% yields optimal results; and (2) a curriculum progressing from simple to complex topics outperforms random training.

• *Code Interpreter + LLM Prompting for Code Synthesis.* Current code generation models rely heavily on large teacher models (e.g., GPT-4) to generate synthetic training data, leading to poor scalability, high costs. And most datasets focus on direct code completion or text-to-code translation, but lack Input-Output (I/O) case-based reasoning tasks (e.g., inferring code from example mappings like "hello" → "olleh"). This gap results in weak generalization for inductive programming challenges.

To bridge this gap, Case2Code [344] generates training data through four steps: (i) Extract executable Python functions (with input/output parameters) from open-source repositories; (ii) Use lightweight LLMs (e.g., InternLM2-7B) to analyze function logic and generate diverse input samples; (iii) Execute functions to obtain real outputs and filter invalid results; (iv) Convert I/O pairs into natural language prompts with diversified templates for improved generalization. This method leverages "code interpreter + lightweight LLM" to cost-effectively produce 1.3M training samples, eliminating reliance on expensive teacher models.

• *LLM-based Clustering for Synthetic Data Evaluation.* In order to study the impact of diversity of large-scale synthetic data, [92] introduces an LLM-based clustering method to quantify synthetic data diversity and analyze its impact on model performance. (i) Builds hierarchical topic trees from web-crawled data via GPT-4 (e.g., Quantum Computing → Qubit Types → Superposition); (ii) Generates diverse datasets by varying topics, prompts (styles, target audiences,

etc.) and LLMs (GPT-4o, Llama-3, etc.). Experiments across different diversity combinations show synthetic data diversity positively correlates with model performance on benchmarks like HellaSwag [449] and ARC-Challenge [142].

• *LLM Prompting for Multimodal Image-Text Synthesis.* Current approaches for synthesizing multimodal pre-training data typically employ two main approaches: (1) the generation of images conditioned on textual input using text-to-image models, and (2) the augmentation of uncaptioned or simple-captioned source images via multimodal models. In the domain of text-to-image synthesis, current methods use diffusion models [145] for image generation. Examples include DiffuseMix [185], which enhances datasets by augmenting image samples through the blending of original and diffusion-generated images, and EDA [387], which applies diffusion models to produce variations of real images that retain semantic consistency while augmenting the dataset. Concerning image captioning, several studies focus on improving the quality of image-text pairs. LaCLIP [133] uses ChatGPT to rewrite existing captions, thereby introducing greater diversity in linguistic expression while maintain the core semantic content. A limitation of this method is the potential for visual semantic loss due to the language model's lack of direct access to the image. To mitigate this, VeCLIP [222] incorporates a multimodal LLM (LLaVA) to provide a detailed visual description of the image contents (e.g., color and shape attributes, objects, and relations among objects). This description is then fused with the original caption by a LLM to yield a more comprehensive final caption. To simultaneously synthesize both image and text samples, CtrlSynth [83] proposes a system comprising three modules: the Florence-large [418] vision tagging model to extract basic visual elements of an image (e.g., color and shape attributes, objects, and relations among objects), the Qwen2-7B-Instruct [434] language model to generate synthetic text which meets the requirements in the instruction, and the stable-diffusion-x1-base-1.0 [314] text-to-image model to generate novel and diverse image samples based on text prompts.

SFT Data Augmentation. The SFT stage of LLM training mainly focus on improvement of specific domains (math, medicine, etc.), aligning LLM's knowledge to instructions, enhancing reasoning ability, etc. Current methods take LLMs as the main method to generate data with some designed frameworks. Many works [179], [260], [290] take existed datasets as seeds to synthesize mimic datasets.

• *LLM-based Knowledge and Q&A Pairs Synthesis.* To enrich or enhance the diversity of data for better model performance, there are various prompt frameworks such as building topic taxonomy [233] and iterative synthesis [179].

For example, to cover various domains of human knowledge, GLAN [233] introduces a knowledge-classification framework for synthetic text generation by GPT-4. (i) Organize knowledge domains (natural sciences/humanities) into disciplines (math/programming) by; (ii) Develop course outlines with units (e.g., "Intro to Calculus") and core concepts (e.g., "Limits"); (iii) Use GPT-4 to create diverse questions by combining concepts, then generate answers with faster GPT-3.5. This structured approach ensures systematic coverage of knowledge areas while balancing generation quality and efficiency.

Though this could enhance understanding of LLM about many domains, but to get better enhancement still needs to focus on one aspect, like math, KPDDS [179] identifies mathematical problem themes (e.g., algebra, geometry) and core skills (e.g., factoring) using GPT-4, then constructs a matrix mapping theme co-occurrence probabilities to guide logical problem generation. GPT-4 synthesizes new questions based on these themes and solutions, which are evaluated for quality (clarity, coherence) and refined via GPT-4 voting. The method further diversifies questions through variations and applies iterative voting to optimize output. This structured approach ensures contextually coherent, avoiding random combinations.

Instead of combining elements like KPDDS (e.g., combining algebra and geometry to synthesize problems), MMIQC [260] enhances mathematical reasoning by iteratively generating complex, diverse problems from existing ones for fine-tuning. Using a seed dataset, GPT-4 creates problems via added constraints, variables, or extended reasoning. A filtering mechanism ensures logical consistency, problem-solution alignment, and correctness, with validated data expanding the dataset iteratively.

• *LLM-based Alignment Data Augmentation.* Domain knowledge is one thing, and lead LLM's knowledge align with instruction is another thing that could be done to get better performance through techniques like few-shot prompting.

AgentInstruct [290] uses LLMs to create scalable, diverse Q&A data. GPT-4 converts raw input (text/code) into structured formats (argument passages, API lists) to enable diverse instruction creation. Multiple GPT-4 agents generate varied task instructions and answers following a detailed taxonomy (e.g., reading comprehension, coding tasks). GPT-4 and Claude-3 then refine tasks by adding complexity (e.g., integrating dense context or escalating difficulty), ensuring high-quality, adaptable outputs.

Similarly, SELF-INSTRUCT [401] aligns LLM's knowledge to prompts by generating task instructions and examples: Starting with a small set of manually written seed tasks, a LLM (e.g., GPT-3) is prompted to generate new task instructions covering various task types, such as classification, question-answering, and generation. Next, different strategies are employed to generate inputs and outputs based on the task type. For instance, for classification tasks, possible class labels (e.g., "positive" and "negative") are generated first, followed by inputs corresponding to each label. For open-ended tasks, a question description is generated first, followed by an answer. The generated data undergoes multiple rounds of filtering, including removing duplicates or invalid data and ensuring input-output alignment.

SFT Reasoning Data Augmentation. Synthesize reasoning data (e.g., code, chain of thought) through techniques like Chain-of-thought(CoT), or utilizing verification tools for more rigorous reasoning.

• *Prompting LLM To Math Reasoning With Verify Tool.* Also for math, MUSTARD [178] utilizes mathematical proof tools to get reasoning enhancement. First, fundamental concepts from the field of mathematics are selected as seeds, and GPT-4 generates corresponding problems through two types of solutions: (1) One is a natural language explanation of the reasoning process, and (2) the other is a formal language solution that can be verified (e.g., code compatible with

TABLE 7: Data Synthesis for LLM.

Stage	Category	Methods
Distillation	Reasoning Augmentation	Cot [353]
		Prompt with Tools [496]
	Data Augmentation	Prompt with Multi-Agent [467]
Pre-Training	Data Augmentation	Distillation + Fine Tuning + Prompt [481]
		Prompt [99], [98], [282], [93],[344], [92]
SFT	Data Augmentation	Prompt [233], [179], [260], [290]
	Reasoning Augmentation	Prompt [178], [173], [346]
		Human Label [253]
		Automated Label [399]
		High Quality Reasoning Data [442], [230]
RL	Prompts Optimization	Prompt [401]
	Human Feedback	RLHF [71]
		RLHF By LLM [476]
RAG	Privacy Protection	Prompt [450]

mathematical proof tools). Next, formal solutions are verified using mathematical proof tools to ensure the correctness of the reasoning and answers. For content that fails verification, the model adjusts based on feedback and re-verifies until a correct result is generated.

• *CoT Data Synthesis By LLM Exploring.* Works mentioned above highly rely GPT-4 for its advanced ability for math to generate problems and solutions to fine-tune for higher reasoning ability. While more recent research try to enhance LLMs' reasoning ability by technique like Chain-of-Thought (CoT, which let LLMs use tokens to output their reasoning steps) and synthesis or label finer reasoning data for training.

By generating CoT data that covers a wide range of reasoning paths through a trial-and-error self-verification loop, [173] breaks the traditional limitation of relying solely on correct reasoning paths. Specifically, multiple LLMs (e.g., Qwen-7B, Llama-3-8B) are utilized to generate diverse solutions for the same mathematical problem (20-50 responses per problem) to encourage models to explore incorrect paths (e.g., wrong formulas, logical leaps) while retaining complete error analysis. Then a verifier LLM (e.g., GPT-4) performs critical analysis on each response: (a) For incorrect paths, annotate the error steps and generate correction suggestions (e.g., "Step 3 misapplies the cosine theorem, which should be replaced with the Pythagorean theorem"). (b) For correct paths, extract key reasoning steps to form a concise CoT. Merge corrected incorrect attempts with correct paths to construct multi-branch CoT.

Similarly, Satori [346] introduces Chain-of-Action-Thought (COAT), a reasoning framework with meta-action tokens (Continue / Reflect / Explore) enabling dynamic pauses, logic verification, and strategy shifts with a two-stage pipeline: (i) Multiple LLM agents generate COAT-formatted reasoning chains to fine-tune a base model for COAT-formatted syntax mastery. (ii) Partial rollbacks ($\leq$5 steps) from historical reasoning (correct/incorrect paths) append <reflect> to trigger revised reasoning with reinforcement learning (RL) combined with rewards for answer correctness, error correction, and penalties for failures. The RL-enhanced model is distilled into base models (e.g., Llama8B) for iterative refinement.

These works propose framework by letting LLM reason by themselves, and we also have works that label reasoning data for fine tuning to get reasoning ability.

• *Reasoning Data Labeling.* [253] compares the effects of outcome supervision (provides feedback based solely on the correctness of the final answer) and process supervision (provides feedback for each step in the reasoning process) on mathematical reasoning tasks by comparing manually labeling the reasoning steps generated by GPT-4 with outcome supervision. The results showed that process supervision model achieved significantly higher problem-solving accuracy (78.2%) compared to outcome supervision model (72.4%)

But this would cost too much manual effort, so MATH-SHEPHERD [399] proposes a method to automatically generate process-annotated data for training Process Reward Models (PRM, which evaluate the quality of each reasoning step). First, complete the remaining reasoning and answers multiple times for the initially generated reasoning steps with LLM, then each step is scored based on two metrics: (1) Hard Estimation (whether the correct answer is generated, with values of 0 or 1). (2) Soft Estimation (the proportion of correct answers generated through this step). These scores assess the step's ability to derive the correct answer.

• *High Quality and Well Format Data Are The Keys To Better Reasoning.* Moreover, LIMO [442] and [230] state that high quality and well-formatted reasoning data are keys to high performance. [442] emphasizes stimulating complex reasoning capabilities in LLMs through a small number of high-quality training examples with questions and reasoning chains. Powerful models (such as R1, DeepSeek-R1-Distill-Qwen32B) are used for evaluation and synthesis, retaining problems that remain challenging. Each problem is accompanied by detailed solutions and reasoning chains (from official solutions, expert solutions, and LLMs-generated Cot, etc.) and filtered by rules-based and LLM-assisted methods.

[230] finds that the overall structure of the reasoning steps is more important than the specific content. With problems from Numina-Math [235] etc. and long CoT generated by DeepSeek-R1 [162] and QwQ-32B-Preview [379] as data to fine-tune. With modification of the fine-tune data, reveals that training the model with incorrect answer samples results in an accuracy drop of only 3.2% compared to training with correct samples. However, shuffling 67% of the reasoning steps in the training samples leads to a 13.3% drop in accuracy on AIME 2024 problems relative to training with correct samples.

Reinforcement Learning The RL stage of LLMs find the most human-preferential responses within the multiple responses generated by LLM of one instruction. Works like [71], [476] manually label the responses or let LLMs do the job.

Label better LLM's response by human or LLMs. To align the model's responses with human expectations, [71] gathers helpful and harmless data through open-ended conversations. Then, a preference model is trained to score the responses in the data, providing a basis for reward optimization in reinforcement learning. The preference scores guide the optimization of the language model's responses. Next, the latest model generates new data, continuously updating the preference model to improve performance on high-quality data. To improve efficiency, [476] proposes a new chatbot evaluation method using language models as "judges" to compare and score chatbot responses, with the goal of automating the evaluation process and reducing human involvement. It introduces two benchmarks: one focusing on multi-turn conversation performance and another collecting user preferences via crowdsourcing. The method also addresses potential biases, such as preferences for answer order or length, through

strategies like swapping answers, using few-shot examples or Chain-of-Thought. The approach demonstrates that language models can achieve high consistency with human evaluators, providing a scalable and interpretable framework for efficient chatbot assessment.

Retrieval-Augmentation Generation. The RAG stage mainly offers knowledge and documents from outside to avoid additional training cost. Main works in this stage of data synthesis focus on privacy issues.

Replace sensitive data with synthesis data. In order to mitigate the privacy issue, [450] proposes a two-stage synthetic data generation and privacy-enhancing method for the RAG stage of LLM.

In the first stage, key information is extracted from the original data (such as "symptom description" and "treatment plan" in medical dialogues), and LLM is used to generate synthetic data based on key information but does not contain sensitive details.

In the second stage, LLMs are applied to the synthetic data, and rewriting strategies are employed to eliminate potential privacy leaks (such as removing specific names or obfuscating descriptions).

This process of evaluation and rewriting is repeated to ensure that the generated data retains its key utility while completely avoiding privacy concerns.

2.3.7 End-to-End Data Processing Pipelines

With above data processing methods, we separately introduce existing frameworks that support common processing operations; practices of integrating some of these methods within pipelines in real-world LLM data preparation; together with some preliminary pipeline orchestration methods.

Principles

When designing data processing pipelines, several critical factors must be considered: (1) the trade-off between data quality and quantity; (2) dependencies across the processing operations (e.g., text extraction necessarily preceding operations like deduplication and filtering); (3) efficiency optimization (e.g., conducting computationally intensive steps like model-based filtering after lightweight processing steps like URL filtering).

2.2.7.1 Typical data processing frameworks

Data processing frameworks provide built-in libraries, operators, and intuitive interfaces that can benefit the design of data processing pipelines for different LLMs. Here we showcase three typical data processing frameworks.

(1) Data-juicer [90] is an open-source framework designed for customizable, high-quality, and efficient data processing. It offers a diverse range of pre-built data processing operators such as data formatting, mapping, filtering, and deduplication. Additionally, the framework features visualization and automatic evaluation, enabling users to receive immediate feedback on their data pipeline. To manage large-scale datasets effectively, Data-juicer is optimized for distributed computing, ensuring robust performance and scalability.

(2) Dataverse [305] is an open-source framework designed to simplify custom ETL (Extract-Transform-Load) pipeline development through an easy-to-use block-based interface that enables users to easily customize by adding, removing, or rearranging blocks. The platform offers a diverse range of pre-built data processing operators, including deduplication, decontamination, bias mitigation, and toxicity reduction, while also supporting the integration of data from multiple sources. Similar to Data-juicer, Dataverse integrates with Apache Spark for distributed processing and supports AWS integration for cloud scalability.

(3) [368] introduces a data processing framework that allows users to customize data processing pipelines using a comprehensive suite of operators categorized in two main modules: (1) The processing module consisting of data reformatting (read and import strctured data), cleaning (removed undesired data such as HTML tags and translate text), filtering, and deduplication (using MinHashLSH in Section 2.3.2) operators; (2) The analyzing module featuring refined data probing and automatic evaluation.

2.2.7.2 Typical data pipelines

Data processing pipelines aim to orchestrate a subset of data processing operations (in a specific order) that transform raw data into high-quality LLM training data (mostly for the pre-training stage). Here we showcase three representative pipelines.

• The MacroData Refinement (MDR) pipeline is designed to construct the RefinedWeb Dataset, which has been used for pre-training Falcon LLMs [311]. MDR refines web-scale data from Common Crawl [11] through three main operations.
(i) Data acquisition: MDR first applies a lightweight URL filter to exclude irrelevant links before any computationally intensive steps. It then extracts text from WARC files using warcio and Trafilatura [73], followed by language identification (i.e., removing content with limited natural language) using fastText [199] as implemented in CCNet [410].
(ii) Data filtering: To eliminate low-quality content, MDR employs both (1) document-level filtering [328] and (2) line-level filtering, which removes noisy content such as social media counters or navigation links.
(iii) Data deduplication: Despite prior filtering, substantial content duplication remains, which can degrade model performance. MDR performs both *fuzzy deduplication* using MinHash and *exact deduplication* with suffix arrays to minimize redundancy. To address computational limits, the Common Crawl corpus is partitioned into 100 segments, with deduplication performed per segment. Additionally, to avoid cross-part redundancy, URL-level deduplication is applied by excluding URLs already retained in earlier segments.

Overall, MDR follows three core design principles: *(i) scale first*, by maximizing data volume from Common Crawl to support large model training; *(ii) strict deduplication*, as rigorous redundancy elimination is critical for training efficiency and generalization; and *(iii) heuristic filtering*, favoring rule-based filters over ML-based ones to reduce bias and maintain transparency.

• The DCLM-Baseline pipeline also processes data from the Common Crawl dataset. Different from MDR, in addition to text extraction and language identification, it applies efficient heuristic filtering [311] to exclude irregular content

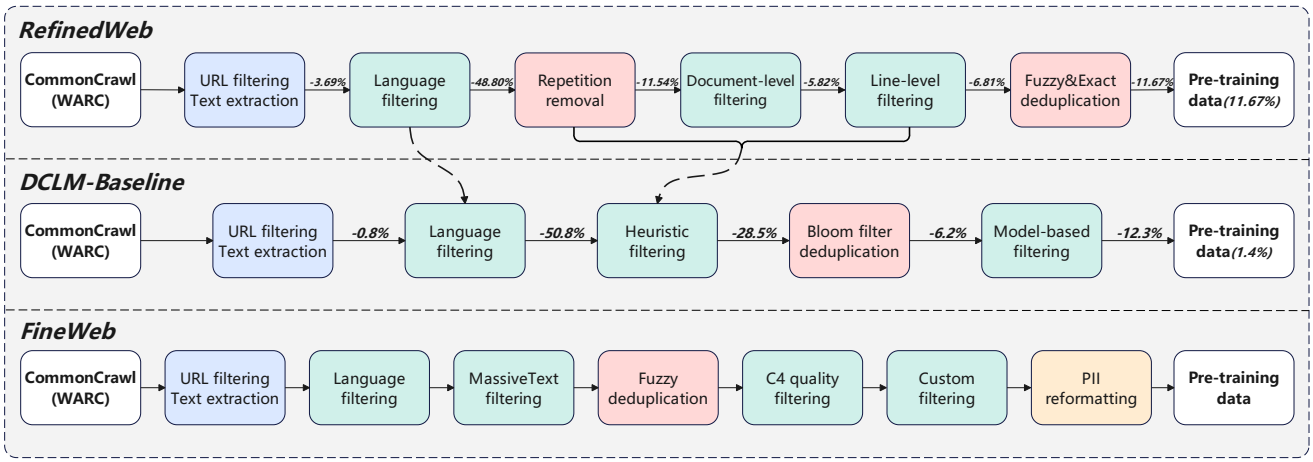

Fig. 6: Typical data processing pipelines for LLMs.

(e.g., toxic words or webpages from illegal sources). Next, DCLM-Baseline adopts a Bloom filter for data deduplication, ensuring its scalability with large datasets. Finally, over the processed data with much smaller size, it conducts model-based quality filtering (most computationally intensive) to remove low-quality content. Specifically, a fastText classifier trained on instruction-formatted data, including OH-2.5 (OpenHermes 2.5) and ELI5 (ExplainLikeImFive), is used to retain the top 10% of documents.

• The FineWeb pipeline (for preparing a 15T-token pretraining dataset) starts with text extraction from WARC files using Trafilatura [73], which is more custom than directly using WET format data and language filtering with fastText. Different from the above pipelines, it conducts MassiveText filtering, i.e., heuristic quality filters and repetition filters on paragraph, line, and gram level [328]. Besides, it conducts fuzzy deduplication using individual MinHash deduplication for each CommonCrawl snapshot, as this approach matches RefinedWeb's performance, whereas global deduplication yields little improvement over non-deduplicated data. After deduplication, given the observation that the C4 dataset yields superior performance on some benchmarks despite its smaller size, a selection of C4 [330]'s heuristic filters is applied to drop low-quality content such as unpunctuated lines and policy statements. Finally, to further enhance data quality, additional custom heuristic filters are developed through a systematic process. Moreover, personal identifiable information (PII) such as email addresses is anonymized using regex patterns in the public release of the dataset.

Compared to MDR and DCLM-Baseline, the FineWeb pipeline is considerably more complex due to its integration of multiple layers of filtering, each inspired by empirical evaluations and comparisons with other datasets such as C4 and RefinedWeb. Its design reflects a trade-off that prioritizes performance over simplicity.

2.2.7.3 Orchestration of data pipelines

The above data pipelines are mostly designed by experience. Instead, Data-Juicer Sandbox [91] proposes a "Probe-Analyze-Refine" workflow, which involves systematically exploring the impact of various data processing operations and their orders on model performance, combining effective operations into data recipes, and optimizing data utilization through duplication analysis and diversity analysis. The orchestrated pipelines are validated through applications on state-of-the-art models like Mini-Gemini (for image-to-text generation) and EasyAnimate (for text-to-video generation).

2.4 Data Storage for LLM

In this section, we introduce storage techniques for LLMs, which we categorize accroding to the tasks they address, including (1) data formats, (2) data distribution, (3) data organization, (4) data movement, (5) data fault tolerance, and (6) KV cache.

2.4.1 Data Formats

Data formats are file formats for training data and models. For LLMs, appropriate file formats for data and models can enhance storage efficiency, accommodate multimodal data, be suitable for model training, ensure security, and influence compatibility across different frameworks.

Principles

Compared to traditional machine learning, LLMs place greater demands on data being multi-modal and in a unified format. The main challenge is how to achieve high data reading efficiency in multi-modal scenarios. Current methods address this using techniques like sequential storage.

Training Data Format. For training data, file formats are required to have good storage efficiency (e.g., TFRecord [44]), be adaptable to large amounts of data (e.g., MindRecord [40]), and sometimes be suitable for model training (e.g., `tf.data.Dataset` [43]).

(1) Pure-Text Formats. Common formats such as CSV, JSON, TSV, and TXT are often used to store pure-text LLM data (though they are not limited to such content). However, for large-scale training datasets (at the PB scale), these formats incur significant storage overhead due to the lack of compression (e.g., not supporting binary encoding), leading to storage waste and slow data loading during LLM training.

To address these issues, TFRecord [44] is based on Protobuf (a highly efficient binary serialization protocol) and stores data in a row-based format. As a binary format, its size is significantly smaller than JSON or CSV. Besides, data can be written and read in a streaming manner, making it especially suitable for scenarios like training where data is consumed sample by sample.

(2) Multimodal Formats. Pure-text formats are not well-suited for multimodal datasets containing images, videos, and text. To address this, file formats such as TFRecord [44] in TensorFlow and MindRecord [40] in MindSpore have been developed to natively support efficient multimodal data storage.

• Unlike traditional formats (e.g., COCO JSON [10], which store image metadata in separate JSON files), TFRecord [44] allows users to encapsulate images, labels, and metadata within a single `tf.train.Example`, eliminating the need for separate label files. Moreover, as multimodal datasets substantially increase data volume, TFRecord supports data sharding, enabling the creation of distributed files that can be assigned across multiple servers to facilitate parallel training.

• MindRecord organizes data into two types of files: (*i*) the data file, which contains a file header, scalar data pages (e.g., image labels and filenames), and block data pages (e.g., image and text) to store training data; and (*ii*) the index file, which maintains indexing information based on scalar data to support efficient retrieval and dataset analysis.

(4) Tensor Data Formats. Compared to the storage formats mentioned above, tensor formats represent data as multi-dimensional arrays. On GPUs or TPUs, such multi-dimensional structures can be partitioned and processed in parallel, making them highly suitable for large-scale computation. For example, `tf.data.Dataset` [43] can organize various raw data types (e.g., images, text) into a unified tensor format, ready for direct use by models. However, tensor formats, due to their dense multi-dimensional storage, incur large storage overhead and offer poor readability, and are typically adopted only in model training.

Model Data Format. Model storage formats need to pay attention to security (e.g., Safetensors [85]) and are usually closely tied to their respective model training frameworks [32], [42], [22].

• Pickle (.pkl [13]) is a Python-specific format supported by almost all Python frameworks and can store any Python object, not limited to model parameters, making it convenient for saving model states and other custom information.

• Safetensors [85] was introduced by Huggingface to address the security concerns inherent in Python's Pickle-based serialization. While Pickle serializes both the data and behavior of Python objects—enabling arbitrary code execution during deserialization—safetensors avoids this risk by focusing exclusively on tensors and their associated metadata. This design ensures safe deserialization without the possibility of executing malicious code. Additionally, safetensors supports memory mapping (mmap), which significantly enhances the efficiency of model loading.

• PyTorch-specific formats (e.g., `.pt`, `.pth` [32]) are optimized for model storage. Typically, `.pth` files are used to save training checkpoints, including model parameters, optimizer states, and epoch information, while `.pt` files are used to store only the model parameters.

• TensorFlow offers two common saving formats [42]: (1) `SavedModel` format for saving the entire model, including computation graph, weights, optimizer; (2) `.ckpt` for storing model weights, optimizer states, and training metadata, and is used to save and restore progress during training.

• ONNX [27] is a cross-framework deep learning model format that supports interoperability across frameworks like PyTorch, TensorFlow, and Caffe2. It offers cross-platform and cross-framework advantages, but does not store training state information.

• The Hugging Face Transformers library [22] adopts a modular storage design, i.e., model weights are stored in binary `.bin` files, model configurations are stored in `.json` or `.txt` files.

2.4.2 Data Distribution

With the development of LLMs, the scale of LLM training datasets and the number of parameters of LLMs themselves are growing rapidly (e.g., 9.5 PB data form Common Crawl [183], DeepSeek-R1 [162] has 617B parameters). A single node cannot store such large-scale data, and the data needs to be distributed across multiple nodes. The key technologies involved mainly include (1) distributed storage systems and (2) heterogeneous storage systems.

Principles

Compared to traditional machine learning, the data (e.g., training data and model data) used in LLMs including both is growing exponentially. The main challenge lies in how to efficiently store and manage such large-scale data. Current approaches address this through distributed and heterogeneous storage systems.

Distributed Storage Systems. Distributed storage systems refer to storing a large-scale datasets across multiple nodes (e.g., JuiceFS [16], 3FS [15]). Traditional distributed file systems (such as HDFS [79]) often come with high costs. Moreover, most distributed file systems still use the POSIX protocol when loading the training data for LLMs, which bring about significant software overhead.

JuiceFS [16], a typical distributed file system based on object storage, uses object storage (e.g., S3 [4]) as the backend to store data. Compared to traditional distributed file systems (file or block storage), distributed file systems based on object storage enables simpler horizontal scaling. It does not need complex directory hierarchy (File Storage) and does not involve complex management logic (Block Storage), thereby significantly reducing storage costs (approximately 20% of the cost of traditional file systems).

As shown in Figure 7, 3FS [15] employs a large number of SSDs for distributed data storage and uses the CRAQ algorithm to ensure data consistency. Specifically, a piece of data is saved as multiple same chunks, which together form a Chain. For read requests, they can be sent to any chunk in the Chain, and the chunk will return the data. For write requests, the writing operation is carried out sequentially on each chunk. When a certain chunk malfunctions, instead of using the incremental data generated during the abnormal

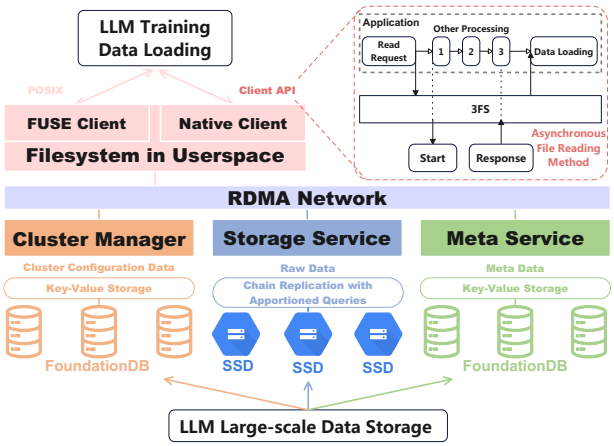

Fig. 7: The storage architecture of 3FS [15].

period to overwrite the data as in traditional methods, it first moves the chunk to the end of the chain. Only when the chunk returns to normal will the entire content of other samples be copied to the abnormal chunk. These operations, while ensuring data consistency, will cause a certain delay in write operations. However, they have almost no impact on read operations, which are more important for LLM training.

Meanwhile, 3FS [15] discovers that in the context of LLM training, the File Cache significantly consumes system memory, thereby degrading overall I/O performance. To address this, 3FS adopts an asynchronous data loading approach, disables file caching and exclusively utilizes Direct I/O for data access, significantly reducing memory pressure. Moreover, it performs system-level alignment of buffer pointers, offsets, and lengths to satisfy Direct I/O requirements, thereby avoiding additional memory copies caused by user-side alignment operations.

Heterogeneous Storage Systems. Heterogeneous storage systems refers to deploying the model state across diverse storage media (e.g., GPUs, CPUs, NVMes Memory). When deploying the model, The Zero Redundancy Optimizer (ZeRO) [333] deploys model states across multiple GPUs. However, simply distributing the model across multiple GPUs often significantly increases computational costs.

Some methods [334], [337], [336], [435] alleviate GPU memory pressure by storing data in host memory or NVMe SSD. vDNN [337] utilizes a per-layer memory management approach based on a sliding window that dynamically allocates memory at runtime based on the computational demands of the current layer. Its memory transfer mechanism includes both static and dynamic policies: the static policy offloads feature maps of all layers or only convolutional layers, while the dynamic policy determines which layers and convolutional algorithms to offload at runtime, balancing trainability and performance based on network characteristics. vDNN fully utilizes CPU memory by offloading intermediate feature maps that are not immediately needed and prefetching them prior to backpropagation. ZeRO-Infinity [334] offloads model states to CPU (e.g. activations) and NVMe memory, effectively alleviating the GPU memory bottleneck. To further reduce memory pressure, it introduces a memory-centric tiling technique that lowers the working memory requirements for LLM training, enabling the execution of large operators without relying on model parallelism.

However, both vDNN and ZeRO-Infinity only utilize CPU's memory without leveraging its computational capabilities. In contrast, ZeRO-Offload [336] retains the parameters and forward/backward computations on the GPU while offloading the remaining computations (such as optimizer calculations) to the CPU, thereby harnessing the CPU's computational power.

Unlike the aforementioned methods that often rely on manual parameter tuning (e.g., specifying offloading targets like CPU or NVMe), ProTrain [435] introduces a model- and hardware-aware automated framework. It incorporates a Memory-Aware Runtime Profiler for monitoring real-time memory and compute loads, partitions parameters into persistent (resident on GPU) and non-persistent (offloaded/loaded on demand) chunks based on their usage patterns, and reduces redundant data copying via pre-allocated chunk buffers.

2.4.3 *Data Organization*

Data organization refers to data operations (e.g., content organization in vector-based organization) during the storage stage that are designed to optimize retrieval accuracy and efficiency in RAG systems. When LLM answers questions, issues like hallucination [187] and lack of timeliness often arise. To address these limitations, RAG [228] (e.g., vector-based retrieval and graph-based retrieval) have been introduced. They provide models with real-time, reliable context during inference. And both retrieval methods are based on the relevant data organization operations (e.g., vector-based organization and graph-based organization).

Principles

Compared to traditional machine learning, LLMs require RAG knowledge to access real-time information. The main challenge is how to ensure both the efficiency and accuracy of retrieval. Current methods address this through vector-based and graph-based data organization techniques. However, existing RAG systems still fall short of meeting the high-quality retrieval demands at the enterprise level, where the document scale can reach millions of pages.

Vector-Based Organization Vector-based organization refers to converting data into vector form for efficient retrieval. It processes the original data through multiple stages (e.g., Content Organization, Chunking, Embedding, Compression and Storage).

(1) Content Organization. For the source data, organizing the content can enhance its logical structure, thereby facilitating improved efficiency and accuracy in retrieval. Works like Dense x retrieval [97], APS [172] refine text into independent semantic units, which could be described as the minimal sentence that include all the necessary context information from the original text to express its meanings, and Thread [57] reorganizes documents into logical units, with each unit containing prerequisites, headers, body content, linkers (describing possible paths for next step), and metadata, enabling

a logical and structured representation of the document's content, which significantly enhances the system's logical coherence and processing efficiency especially in complex tasks (e.g., troubleshooting and dynamic operational workflows).

Similarly, [89] organizes the content of scientific papers into a hierarchical tree structure, where the root node of the tree is the paper's title and child nodes are different sections, such as the introduction and methods. The relationship between parent and child nodes represents the global-local content relationships, such as the connection between the abstract and introduction. Then it traverses the paths from the root node to the leaf nodes to extract important contextual information.

(2) Chunking. In vector-based retrieval, embedding long texts may reduce retrieval efficiency. Thus, an effective chunking strategy is required to divide the text into appropriately sized segments for encoding. The optimal chunk length needs to balance retaining fine-grained semantics and maintaining sufficient context, since a too long text might suffer from significant semantic compression during embedding, while too short a text would increase processing costs.

Allowing overlap between consecutive chunks ensures that important information at the boundaries is not lost and the continuity of context is maintained. Different from traditional chunking, MoG [480] adopts a dynamic chunking strategy, which chunks data when building the knowledge base, where MoG dynamically determines the optimal granularity (e.g., sentence-level, paragraph-level, or section-level) of the knowledge source based on the input query through a trained router. The router, implemented as an MLP, assigns weights to different granularities to guide snippet selection. MoGG [480] extends MPG by converting reference documents into graphs and redefining granularity as hopping ranges, enabling effective retrieval of dispersed information for complex queries.

(3) Embedding. In vector-based retrieval, the original input (text, images, audio, or other domains) is transformed into dense vector representations using models specifically adjusted for each data type. These representations encapsulate the underlying semantic meaning of the original content, and are then stored in a vector database for storage and retrieval. Various embedding models are used to correctly encode semantic information:

• *BGE* uses a bilingual joint training framework that combines language-specific subword tokenization and specialized adaptation layers. This design aligns semantic representations across languages, improving cross-lingual retrieval accuracy [94].

• *STELLA* features a cross-instance attention aggregation mechanism that explicitly captures inter-sentence dependencies during pretraining. Besides the general embedding model, STELLA offers an extra dialogue model in incomplete query situations where the user input has problems such as semantic omission and reference digestion. This reduces the embedding dimensions and inference latency, making it especially effective for large-scale tasks [24].

• *GTE* introduces a dual-negative sampling strategy within its contrastive learning paradigm. Though introducing negative samples usually works in series of embedding models, this strategy incorporates more reverse contrastive terms within a fixed batch, strengthening the model's ability to distinguish subtle semantic differences. [249].

(4) Compression. Vector retrieval in LLMs differs from regular vector retrieval in that semantically similar vectors are often high-dimensional, so dimensionality reduction techniques are needed to reduce storage pressure.

• ***Linear Dimensionality Reduction.*** Locally-adaptive Vector Quantization (LVQ) [50] centralizes the data and scales each vector individually, calculating the quantization bounds adaptively in a localized manner, fully utilizing the quantization range to compress the vectors. This method is typically suitable for compressing vectors with around 100 dimensions, but it performs poorly when the vector dimension is very large, such as tens of thousands.

LeanVec [380] combines linear dimensionality reduction with LVQ for vector compression. In ID(In-distribution) scenarios, LeanVec uses PCA, while in OOD(Out-of-distribution) scenarios, it introduces the LeanVec-OOD optimization method, which minimizes the square of the inner product between the query vector and the representation error to find the optimal projection subspace for both the dataset and the query set, thereby reducing the vector dimension. However, LeanVec is a simple linear dimensionality reduction method, and its performance may be affected in terms of accuracy when reducing the dimensionality drastically.

LeanVec-Sphering [381] modifies the loss function, transforming the problem of finding the projection matrix into an optimization problem under the Mahalanobis distance, which allows for more effective discovery of the optimal projection matrix, thereby better preserving the similarity structure between vectors when processing high-dimensional vectors.

• ***Non-linear Dimensionality Reduction.*** GleanVec [381] uses spherical k-means clustering in the data partitioning stage to group vectors based on direction, capturing the data's structural features. By associating cluster labels with vectors, it narrows the search range and reduces unnecessary calculations during inner product computation. In the local linear dimensionality reduction stage, GleanVec applies the LeanVec-Sphering method to reduce dimensionality within each cluster, preserving the inner-product relationship, which simplifies calculations while maintaining accuracy.

(5) Storage. After the above steps, the data will be stored in vector form in a vector database. During LLM inference, the model vectorizes the input and uses similarity metrics such as cosine similarity or dot product to retrieve the most relevant data from the database.

Faiss [125], when storing vectors, relies on the chosen index type. The Flat Index stores all vectors directly, such as IndexFlatCodes, which stores vectors in a flat array and supports sequential IDs. It is ideal for small datasets with high-precision requirements. The IVF Index clusters vectors with a coarse quantizer and stores them in inverted lists, supporting user ID operations and optionally using a DirectMap for efficient access. This reduces the search range and speeds up retrieval, making it suitable for large datasets. The PQ Index compresses vectors by splitting them into sub-vectors and quantizing them with a k-means quantizer (e.g., PQ6x10), trading accuracy for reduced storage space, making it suitable for high storage demands and lower precision needs.

In the Milvus [26], vector storage differs based on the number of vectors per entity. For single-vector entities, vectors are stored continuously without row IDs. Since vectors are sorted by row ID and have the same length, a vector can be

directly accessed using its row ID, reducing storage overhead and improving query access efficiency. For multi-vector entities, vectors are stored in a columnar format. For example, for entities A and B, each with two vectors, the storage format is (A.v1, B.v1, A.v2, B.v2). This columnar storage enables more efficient data processing by vector dimension, facilitating batch operations and improving processing performance.

Weaviate [34] utilizes a graph data model to manage data entities, storing vectors as node attributes linked to these entities. For example, in the case of text data, vectors generated by a text embedding model are associated with their corresponding text entity nodes, enabling efficient graph traversal and multi-hop queries based on vector similarity. Additionally, Weaviate can store vectors alongside structured attributes. For instance, the vectors of e-commerce products, along with structured attributes such as price and category, are stored in the corresponding entity nodes. This allows for hybrid queries that combine vector similarity and structured attribute conditions, enhancing query flexibility and practicality.

LanceDB [25] uses a columnar storage format called Lance to store data. Compared to traditional Parquet formats, Lance introduces the concept of a table schema. A single row in LanceDB can store images, text, audio, video, and any number of vectors corresponding to different parts of the original data, and it can be dynamically updated. This makes LanceDB particularly suitable for storing multi-modal data. Currently, LanceDB is used for handling various RAG tasks.

Graph-Based Organization. Unlike vector-based organization, which helps LLM find knowledge related to a user's query through fuzzy searching, graph-based data explicitly represents entities and their relationships, enabling the identification of precise matching information in the database. We will introduce graph-based organization from two aspects: indexing and storage.

(1) Indexing. In the indexing phase, it is necessary to establish an efficient indexing architecture to address the issue that directly retrieving raw triples is inefficient for complex queries such as multi-hop reasoning or path search, because the inherent sparsity in the graph structure often leads to significant query latency.

GraphRAG [127] adopts community clustering and hierarchical summarization strategies. It uses the Leiden algorithm to detect tightly connected subgraphs, called communities, in the knowledge graph. Then, it generates hierarchical summaries for each community. Once a certain element in a triple is retrieved, the index collects relevant community summaries and sends them for inference. For example, it can condense hundreds of triples related to "quantum mechanics" into a semantic summary: "Quantum mechanics is the fundamental theory describing the behavior of matter and energy at microscopic scales".

Furthermore, LightRAG [164] integrates deduplication functionality to identify and merge identical entities and relations from different paragraphs. In real-time update scenarios, LightRAG introduces the Delta Index mechanism, which builds local indexes only for newly inserted edges and entities, using background merging threads without the need for community reconstruction, significantly reducing overhead related to community detection compared to GraphRAG.

MiniRAG [136] proposes a semantic-aware heterogeneous graph indexing mechanism, integrating text chunks and named entities into a unified structure, reducing the reliance on large language models for complex semantic understanding. The low semantic calculating requirement while deploying grants MiniRAG a more excellent performance on resource-constrained devices compared to other methods.

(2) Storage. Graph data is usually stored in graph databases in three models: property graph models [292], RDF (Resource Description Framework) models [65], and multi-model [1].

Neo4j, JanusGraph, and TigerGraph use property graph models [292] to store graph-based data. A property graph model consists of "nodes" and "edges," where both can contain attributes (key-value pairs). This model uses query languages like Cypher and GSQL, designed for relationship modeling and querying, making them highly suitable for complex relationship queries during RAG in LLMs.

Amazon Neptune [65] supports both property graph models and RDF models for graph-based data storage. The RDF model, based on triples (subject, predicate, object), represents entities, attributes, and relationships in a way that enhances knowledge reasoning. By combining these two models, Neptune can meet diverse knowledge storage needs, such as rapid queries and deep reasoning.

ArangoDB [1] uses a multi-model approach to store graph-based data. It supports multiple data models (e.g., document, key-value pair, graph), allowing the selection of appropriate storage and query methods depending on the requirements. This allows ArangoDB to store graph data (relationship information), document data (context or factual information), and key-value pairs (configuration or metadata) in the same database, facilitating LLMs to extract relationships from knowledge graphs while also retrieving document-type data (e.g., specific context information).

2.4.4 Data Movement

Data movement refers to the process of moving data from storage nodes to computing nodes. This process can achieve high data movement performance by caching data. Meanwhile, offloading data and operators to multiple nodes for computation can improve the speed of data preprocessing. Additionally, the highest overall performance can be achieved by overlapping data storage and computation operations to jointly schedule storage and computing resources.

Principles

Compared to traditional machine learning, LLMs involve massive data transfers from storage nodes to compute nodes. The main challenge is how to accelerate the data moving rate. Current methods address this through data caching, compute-storage overlap, and data/operator offloading.

Caching Data in advance can increase the data moving rate. However, if a fixed cache policy is used, in order to meet the IO requirements of training, the configured storage capacity often far exceeds that required for storing the dataset [469]. Therefore, a dynamically adjustable cache policy is needed. Some methods [219], [161], [469] dynamically adjust the cache

mechanism by analyzing the characteristics and requirements of LLM jobs in real time.

Quiver [219] optimizes cache sharing strategies based on the following IO characters during model training: (1) data shareability (due to significant overlap in data access within and across jobs), (2) substitutability (the I/O order does not affect job correctness, enabling small caches to improve performance by substituting data and reducing thrashing), and (3) predictability (using mini-batch processing times to estimate job sensitivity to I/O performance for informed cache allocation).

Fluid [161] dynamically adjusts cache capacity according to I/O conditions, optimizing the online training speed for each individual LLM job. Specifically, Fluid uses a coordinator to monitor the processes of LLM jobs. It calculates the number of samples within a specific time window based on the batch sizes fed back by the jobs, and thus obtains the real-time training speed. Subsequently, based on the concept of the TCP congestion control algorithm [315], it adopts a trial-and-error approach to dynamically adjust the cache capacity. When the training speed increases, the cache capacity is increased according to the preset scaling-up factor and scaling step. Conversely, when the training speed decreases, the cache capacity is decreased according to the preset scaling-down factor and scaling step.

Meta proposes Tectonic-Shift [469], a hybrid storage architecture that integrates flash memory with the traditional HDD-based distributed file system Tectonic. Tectonic-Shift organizes data segments into buckets for storage in flash memory and determines segment admission and reinsertion by comparing bucket priorities (computed from both historical and predicted future access patterns) against dynamically adjusted thresholds. It also optimizes the segment size (e.g., 256 KB) of CacheLib [9] to improve flash memory utilization.

Data/Operator Offloading refers to offloading data preprocessing operations such as shuffling, sampling, and augmentation, to multiple devices in order to improve processing speed. Currently, data preprocessing pipelines (e.g., tf.data) are typically performed on the CPU, whose efficiency is often lower than the training speed achieved by Machine Learning (ML) accelerators like GPUs and TPUs. So enhancing the efficiency of data preprocessing to match the high-speed processing capabilities of ML accelerators has become a challenge [159].

Some research [158], [67] offload data preprocessing tasks to remote CPU servers. Cachew [158] divides the input dataset of each job into independent subsets for processing by remote CPU nodes. Additionally, users can specify locations for caching and reusing data in the input pipeline. The scheduler makes decisions during runtime based on specific metrics and algorithms through automatic scaling and caching strategies. The automatic scaling strategy adjusts the number of worker nodes according to client-reported metrics. The automatic caching strategy compares the processing times of different cache locations and selects the optimal caching scheme. The tf.data service [67] addresses input data bottlenecks by horizontally scaling CPU nodes and leveraging a coordinated read mechanism to mitigate straggler issues caused by input size variability in distributed training. Specifically, it is comprised of four key components: a dispatcher, a pool of workers, clients, and an orchestrator. The dispatcher manages dataset assignment to workers using various sharding strategies, for example, the OFF strategy performs no sharding, the DYNAMIC strategy applies disjoint first-come-first-served sharding, and several static sharding strategies are also supported. Workers are responsible for actual data processing. Clients issue data processing requests to the workers. Orchestrator deploys the aforementioned three components as containers within the same Borg [384] unit.

Although the above method of offloading to remote CPU servers can alleviate data stalls, the cost of remote CPUs is high, and the resources of ML accelerator nodes are not fully utilized. Pecan [159] introduces two strategies, *AutoPlacement* and *AutoOrder*, to alleviate input data preprocessing bottlenecks and reduce training costs. The *AutoPlacement* strategy dynamically schedules data preprocessing workers across ML accelerator hosts and remote CPU servers. It first establishes a baseline batch processing time for model training, incrementally adds local workers, and then prunes redundant remote workers to determine the optimal combination of local and remote resources. The *AutoOrder* strategy analyzes the transformation operations within the input data pipeline, reordering them to place data-reducing transformations (such as sampling, filtering, or image cropping) earlier and data-expanding ones (such as image padding and one-hot encoding) later. While adhering to user-specified ordering constraints, this reorganization improves the preprocessing throughput of individual workers.

Different from the works that are only compatible with a single training framework as mentioned above (e.g., Cachew and tf.data service can only work with TensorFlow). Powered by native composable operators (e.g., data loading, transformation, and filtering functions), Cedar [468] can flexibly support different ML frameworks and libraries, enabling users to effortlessly build data pipelines.

Overlapping of storage and computing means that the data loading and computation processes in LLM training alternate. In LLM training, which proceeds in data batches, ideally the data loading unit can prepare the next batch while the computing unit processes the current one, reducing overall training time. However, if a data isn't cached locally, its need to load the data through remote I/O bandwidth. When this bandwidth is insufficient, computation pauses to wait for data loading, creating an IO bottleneck. Some researches optimize the pipeline at different training stages (e.g., the pre-training and SFT stage [466], the RL stage [479]).

SiloD [466] leverages the characteristics of the pipelined execution of data loading and computation at the pre-training and SFT stage to build an enhanced performance evaluator. When data loading becomes the bottleneck, it uses a learned model (IOPerf) to quantify the cache and remote I/O demands of different training jobs,providing support for resource allocation in the pipelined of data loading and computation.

Compared with the pre-training and SFT stages, the RL stage requires an additional training of the reward model to evaluate the output of the original model. This leads to a greater amount of computational resources remaining idle (pipeline bubbles) during the RL stage. RLHFuse [479] takes advantage of the independence between the original and reward models during the training stage to break the training

task into sub-tasks of micro-batches. In the case of differences in the sizes and parallel strategies of the two models, it first transforms the problem to ensure that each stage of the two models uses the same number of GPU resources, and then uses the simulated annealing algorithm [213] to generate a fused pipeline schedule.

2.4.5 Data Fault Tolerance

Data fault tolerance refers to the ability to quickly resume from the point of interruption during model training by storing checkpoints or performing redundant computations in the event of training interruptions.

Principles

Compared to traditional machine learning, LLMs place greater emphasis on fault tolerance during training due to their large model sizes and the high cost of retraining. The main challenge is how to quickly resume normal training in the event of an interruption. Current methods address this by saving checkpoints or using redundant computation.

Checkpoints. Some methods store the model state as checkpoints to handle training interruptions. However, restoring model states across multiple platforms or frameworks may encounter compatibility issues. At the same time, frequently saving model checkpoints can consume a large amount of storage space, especially during large-scale model training.

For compatibility issues, PaddleNLP [29] has developed a unified model storage technology. It stores model weights, optimizer weights, and other data in a unified *safetensors* format, eliminating the need to differentiate distributed strategies during checkpoint storage. Specifically, when the distributed training strategy changes (e.g., switching between data parallelism and model parallelism) or the number of machines is adjusted, Unified Checkpoint enables training to resume using only a single complete checkpoint, without requiring separate checkpoints for each configuration.

(1) Asynchronous Storage. Apart from standardized checkpoint storage, for frequently saving model, some researches [291], [194] aim to accelerate checkpoint saving through asynchronous storage without affecting the model's training speed.

CheckFreq [291] employs a two-stage checkpointing technique designed to capture model state copies in memory for asynchronous storage while ensuring model parameter consistency through pipelining with subsequent iteration computations. Specifically, when idle GPU memory is available, it prioritizes snapshotting on the GPU to reduce costs; otherwise, it stores checkpoints in CPU memory and adjusts the checkpoint frequency accordingly.

In the training of LLMs on the MegaScale system [194], HDFS is used to store the model state. When storing model states, there are problems of *balancing the checkpoint frequency and dealing with the HDFS bandwidth bottleneck* during model recovery in the training process. To address this, MegaScale adopts a two-phase storage approach: (1) GPU worker nodes quickly write the on-chip state to the host memory and continue training; (2) a background process asynchronously transfers the state to HDFS to reduce interference with training. When resuming training, a worker node in the specified data parallel group reads the shared state partition and broadcasts it to other nodes, reducing the HDFS load and alleviating bandwidth pressure.

(2) Hierarchical Management refers to storing model checkpoints across a multi-level storage system, storing the checkpoints that may be needed in the closer storage nodes, aiming to improve recovery speed. Gemini [403] stores checkpoints in a hierarchical storage system composed of local CPU memory, remote CPU memory, and remote persistent storage. It introduces a near-optimal checkpoint placement strategy for CPU memory. By analyzing the relationship between the number of machines and checkpoint replicas, it flexibly adopts group placement or ring placement to maximize the likelihood of recovery from CPU memory in the event of failures. ByteCheckpoint [389] manages checkpoint files using an architecture combining SSD and HDD storage servers. New checkpoint files are stored as "hot" data on SSDs for quick access due to evaluation task downloads after creation. Once the evaluation is completed and there are no training anomalies, their access frequency drops, and they become "cold" data, being migrated to HDDs to free up SSD space and ensure the hot storage can efficiently store currently frequently accessed checkpoint files.

Redundant Computations Unlike checkpoint, some methods [382], [186], [147] are based on parallel computing and redundantly compute the state data of the model, enabling quick recovery of the training state from non-failed nodes in case of failures.

Inspired by the RAID disk redundancy technology [307], Bamboo [382] enables each computing node to perform computations not only on the neural network layers it is responsible for, but also on some layers of its neighboring nodes as redundant computations. When a node is preempted, its predecessor node has all the information required for training, allowing the training to continue without wasting previous computational results.

Unlike Bamboo's node-based redundant computation, Oobleck [186] uses pipeline templates to define training pipeline execution, specifying node allocation, stage numbers, and model layer-GPU mappings. During training, at least $f + 1$ logically-equivalent yet physically-heterogeneous pipelines are instantiated from these templates, considering the fault tolerance threshold f and batch size. When a pipeline node fails, Oobleck leverages other pipelines' model state redundancy and reinstantiates the pipeline to resume training.

Unlike Bamboo and Oobleck, which use pre-set redundant computations in standby, ReCycle [147] leverages the computational redundancy inherent in parallel training to reassign the tasks of failed nodes to nodes with the same processing in other data-parallel groups. This unique approach enables quick resumption of training without the need for spare servers.

2.4.6 KV Cache

LLMs use auto-regressive generation, where each token depends on prior ones. KV Cache avoids redundant computation by reusing stored key-value pairs, improving efficiency. How-

ever, its memory grows with sequence length, making efficient cache management crucial.

Principles

Compared to traditional machine learning, LLMs require KV cache to accelerate inference. The main challenge lies in efficiently managing the cache as the KV size grows rapidly. Current methods address this by indexing KV, shrinking KV, and managing KV placement or cache space.

Cache Space Management refers to separating the logical structure of the KV cache from its physical storage implementation, which facilitates memory allocation and improves memory utilization. vLLM [220] and vTensor [428] divide the KV cache into fixed-size blocks and store them in a non-contiguous manner. vLLM manages these blocks through a mapping mechanism, while vTensor stores the fixed-size KV cache blocks non-contiguously in physical memory. This decouples the logical and physical KV blocks, utilizing a block table to manage dynamic memory allocation by tracking the mapping relationships and fill states.

KV Placement refers to using a perception strategy to store frequently used KV in faster storage media (such as GPU memory), while storing less frequently used KV in slower storage media (such as SSD), or releasing them directly. RAGCache [197] provides a prefix-aware PGDSF replacement policy that prioritizes cache nodes based on access frequency, size, and recomputation cost. And stores frequently accessed data in fast GPU memory and less frequent data in slower host memory, maximizing cache efficiency. CachedAttention [148] leverages the inference job scheduler to observe the jobs waiting for execution. To improve cache efficiency, the KV cache of a pending job is prefetched into the host memory from disk before execution. Meanwhile, KV caches that are no longer required are evicted, based on the jobs waiting to be executed.

KV Shrinking KV Cache Shrinking refers to trimming or reducing the KV Cache in order to lower memory usage and improve inference efficiency. CacheGen [265] uses a customized tensor encoder to encode the KV cache into a more efficient bitstream, thereby reducing bandwidth usage. It also compresses the KV cache using techniques such as block-based encoding, hierarchical quantization, and arithmetic encoding, while dynamically adjusting the compression level and transmission method based on network conditions to ensure low latency and high generation quality.

Unlike CacheGen, which only considers intra-layer redundancy, MiniCache [255] is based on the similarity of KV cache states in adjacent layers. It decomposes the state vectors into magnitude and direction components, calculates the direction vectors using SLERP [354], and merges the KV caches of adjacent layers to form a merged cache that contains information such as direction vectors, magnitudes, and angles.

Compared with the traditional method of storing the complete KV data, HCache [150] only stores the hidden states (the size of the hidden states is only half that of the KV cache, and recomputing the KV cache from the hidden states can reduce the computational load). When restoring the state, a bubble-free restoration scheduler is used to concurrently execute the transmission of hidden states and the recomputation from hidden states, maximizing the overall resource utilization.

KV Indexing refers to the process of constructing an indexing architecture for the KV Cache to accelerate the query process of the KV Cache. ChunkAttention [440] organizes the KV cache into a prefix tree using a prefix-aware KV cache (PAKV), sharing key-value tensors of common prefixes to accelerate the corresponding KV query process. [478] proposes Prefix Sharing Maximization (PSM): By dynamically reordering data columns and rows, it maximizes prefix sharing among requests to improve cache hit rates. Column Reordering sorts columns based on value frequency and size, prioritizing those with more shared prefixes. Row Sorting groups requests with identical prefixes together, further enhancing cache reuse.

2.5 Data Serving for LLM

Data service encompasses data preprocessing operations carried out after data is transferred from storage to computing nodes and before its actual utilization by the LLM, aiming to facilitate more effective data consumption by the LLM. These data preprocessing operations include: data shuffling, data compression, data packing, and data provenance.

2.5.1 Data Shuffling

Data shuffling in data serving means that different data needs to be selected and provided to LLMs at various stages (e.g., in different epochs for pretraining). For example, corresponding training data needs to be supplied according to the training requirements during the training stage; during the RAG stage, corresponding knowledge needs to be supplied based on the degree of relevance to the questions.

Principles

Compared to traditional machine learning, LLM applications are divided into multiple stages, each requiring different types of data to be fed into the model. The main challenge is how to select data that meets the specific requirements of LLMs. In the training stage, current methods provide training data by scoring based on data samples or model states, or by using empirical training strategies. In the RAG stage, data is selected through metrics, rules, or models to supply relevant knowledge to the LLM.

Data Shuffling for Training. As LLMs continuously trained over new tasks, it may begin to lose its ability to retain early task knowledge, a phenomenon known as catastrophic forgetting [287], [286]. To address this, some data supply methods are employed to manage datasets during the training process and provide high-quality data. Meanwhile, some methods, instead of altering the dataset, propose reasonable learning strategies.

(1) Data Pruning. Data pruning refers that during the training process, partial shuffling is carried out on the training dataset, and high-quality data is retained, so that the model is trained on the data that has not been fully learned and is of high quality.

Sample Scoring. Some methods [137], [66] prune datasets by scoring samples, selecting high-scoring samples for subsequent training. [137] applies the EL2N metric to identify important examples in a dataset, written as $\chi(x_i, y_i) = \mathbb{E}\|f(x_i) - y_i\|_2$, where $f(x_i)$ is the model's prediction and y_i is the true sample. Based on the computed EL2N values, it periodically prunes irrelevant data during training. [66] extends the EL2N metric to evaluate sample importance, written as $\hat{\chi}_{ema}(x, y) \leftarrow \alpha \cdot \hat{\chi}_{nlu}(x, y) + (1 - \alpha) \cdot \hat{\chi}_{ema}(x, y)$, where α is a smoothing parameter. Based on extended EL2N values, it periodically selects data subsets for training.

Model State Scoring. Unlike the aforementioned approach of scoring samples and prune the dataset, some methods [372], [56], [416], [276] prune the distribution of dataset by scoring the model's state (such as training loss and learning status).

Moving-one-Sample-out (MoSo) [372] identifies and selects the most informative LLM pre-training samples by assessing the influence of a specific sample on the training loss. The MoSo score measures how the training loss over the dataset $\mathcal{S}$ excluding z (i.e., $S \setminus z$) would change when the sample z is removed. This approximation measures the agreement between z and $S \setminus z$, where the sample is considered important and receives a higher score if the gradient of z is consistently aligned with the average gradient.

Similarly, Velocitune [276] is a dynamic domain weight adjustment method based on learning velocity, which is defined as $V_t[i] = \frac{\ell_t[i] - \ell_{\text{target}[i]}}{\ell_{\text{init}[i]} - \ell_{\text{target}[i]}}$, where $V_t[i]$ denotes the learning velocity for domain i at step t, $\ell_t[i]$ is the current loss for domain i, $\ell_{\text{target}[i]}$ is the target loss for domain i, predicted by the scaling law [201], $\ell_{\text{init}[i]}$ is the initial loss for domain i, calculated before training starts. The method calculates the learning velocity of each domain and dynamically adjusts the sampling weights, giving more attention to domains with slower learning progress, thereby achieving a balanced learning effect.

Some methods [56], [416] combine reinforcement learning based on scoring the model to adjust the dataset. ODM [56] is based on the multi-armed bandit algorithm. It regards each data domain as an arm and uses classical reinforcement learning methods. By taking the training loss as the reward function, it optimizes the data mixing ratio online to adapt to training dynamics. That is, it dynamically adjusts the sampling weights of each data domain and preferentially selects data with high information gain and large losses.

MOS [416] proposes a *scoring network* that dynamically adjusts the sampling probabilities of different datasets based on the model's current learning state, combined with reinforcement learning, to alter the distribution of training data. This adjustment is guided by three reward functions: (*i*) *Transferability* for measuring the similarity (e.g, cosine distance) between datasets as the reward. (*ii*) *Learning difficulty* for measuring the perplexity changes. (*iii*) *Learning trajectory* for smoothing the reward values using Exponential Moving Average (EMA) to more stably optimize the sampling distribution.

(2) Training Strategy. In addition to directly prune the dataset during training, appropriate learning strategies can also alleviate catastrophic forgetting. [123] found that different abilities vary with data volume, with mixed data improving abilities at low resources and causing conflicts at high resources. Thus, DMT [210] is proposed, which first fine-tunes on a specific dataset and then fine-tunes on mixed data to effectively balance general and specialized abilities and mitigate conflicts and forgetting. It proposes a strategy where training data are sorted based on criteria like input length, attention weights and training loss, allowing the model to gradually learn from simple tasks to more complex ones.

Data Selection for RAG. In the RAG stage, it is necessary to retrieve the stored knowledge (see details in 2.4.3) and provided the retrieved results to the LLM. During this process, it needs to ensure the effectiveness of the retrieved results in order to obtain better answers from the LLM [280]. Currently, the retrieval quality is mainly guaranteed through RAG knowledge filtering and RAG knowledge re-ranking.

(1) RAG Knowledge Filtering. RAG knowledge filtering refers to filtering out documents with poor relevance after retrieval. Some methods [280], [114], [87] use a model as a judge to filter documents. [280] uses small language models (SLMs) as filters, performing preliminary predictions and evaluating difficulty. For easy samples, the SLM's predictions are used as the final decision; for difficult samples, the top N most likely labels are selected from the SLM's predictions for subsequent re-ranking. In Chatlaw [114], after retrieving relevant information, the LLM evaluates the retrieved content. Only content that is deemed highly relevant after evaluation is used to generate the final response, effectively reducing interference from irrelevant or incorrect information. MAIN-RAG [87] collaboratively filters and scores retrieved documents by leveraging multiple LLM agents to enhance relevance and reduce noise. The framework adopts a dynamic filtering mechanism that uses score distributions to adjust relevance thresholds, ensuring high recall of relevant documents while minimizing computational overhead.

(2) RAG Knowledge Re-ranking. After filtering, multiple documents may remain, requiring re-ranking of the retrieval results to place the most relevant ones at the top for more accurate model output. Research on [128] shows that using a large model for re-ranking performs better than methods like Maximum Marginal Relevance (MMR) and Cohere re-ranking. For large model re-ranking, general-purpose large language models (e.g., GPT) can be used directly, or specialized zero-shot re-ranking models such as Cohere rerank [12] or RankVicuna [318] can be employed. The latest ASRank [47] leverages pre-trained LLM to compute the matching probability between document answers and answer cues, scoring and re-ranking the retrieved documents.

2.5.2 Data Compression

Data compression refers to compressing the input data for the model. Previous studies have shown that prompts are crucial for triggering LLM domain-specific knowledge, and prompts are typically designed based on specific tasks (including chain-of-thought, context learning, and historical dialogues). As the complexity of chain-of-thought, context learning, and RAG increase, longer prompts are required [189]. However, overly long prompts may lead to higher response latency, increased costs, and even exceeding the maximum token limit. Existing methods mainly compress the model inputs in two aspects. Some methods [427], [101], [348], [200], [335] compress the retrieved results in the RAG stage and then put them into the prompt, while other methods compress the entire prompt [189], [190], [303], [293], [102].

Principles

Compared to traditional machine learning, LLMs often require longer inputs, and in some cases, the input must be compressed to fit into the model. The main challenge is how to compress the input without losing important information. Current methods mainly achieve this through compression based on information entropy, rule-based templates, or model-driven approaches.

RAG Knowledge Compression The retrieved RAG knowledge can be compressed by a model to make small texts carry more information. Techniques like RECOMP [427], CompAct [348], and FAVICOMP [200] adopt rule-based RAG context compression schemes, where predefined rules or templates explicitly guide the model to extract key information and remove redundant content. Alternatively, methods like xRAG [101] and COCOM [335] use soft prompt-based RAG context compression schemes, where learnable parameters (such as the modality projector W in xRAG or the overall model training in COCOM) enable implicit vector learning. These implicit vectors dynamically adjust attention weights when the model processes input, allowing the model to adaptively optimize context representations under context compression.

Prompt Compression. Prompt compression means that after the retrieved knowledge is put into the Prompt, the entire Prompt will be compressed.

(1) Metric-Based Compression. Some studies [189], [190], based on the hypothesis that a vast amount of knowledge is stored in the model parameters, have proposed methods to compress prompts while minimizing information loss. LLMLingua [189] uses a perplexity criterion to remove redundant tokens from the original prompt. By quantifying the negative logarithmic probability (perplexity) of each token through a small model, LLMLingua identifies and removes tokens that can be predicted from the model's inherent knowledge, thereby shortening the prompt while retaining essential context.

LLMLingua's extended version, LongLLMLingua [190], uses a dual-granularity compression strategy: (*i*) Coarse-grained compression initially filters key information at the document level to provide more focused content for fine-grained compression; (*ii*) Fine-grained compression further optimizes at the token level to precisely retain key information. These two strategies work together to improve the quality of the prompt and model performance. LongLLMLingua also assigns different "compression budgets" to documents based on their importance, aiming to achieve the best global compression effect.

(2) Finetuned-Model-Based Compression. Unlike the aforementioned methods that use a small model's perplexity for compression, some methods [303], [293], [102] directly perform the compression task end-to-end by fine-tuning a model. LLMLingua-2 [303] defines prompt compression as a problem of classifying tokens and trains a dedicated model for compression. It uses a Transformer encoder to capture bidirectional contextual information, ensuring that the compressed prompt is faithful to the original. [293] proposes a technique called 'gisting', where a language model is trained to condense the prompt into a compact 'gist token'. These tokens encapsulate the core semantic content of the prompt and can be cached for later use. This method achieves a compression rate of up to 26 times. [102] suggests a method to transform pre-trained language models into AutoCompressors. The AutoCompressor compresses long contexts into summary vectors, and training is performed on the model parameters using these summary vectors.

2.5.3 Data Packing

Data Packing aims to address the requirement for uniform sequence lengths in LLMs' training inputs, which combines short texts in an appropriate way to enhance text coherence and reduce the number of padding tokens. In this way, we can avoid the excessive truncation caused by the drawbacks of simple concatenation and splitting methods [116].

Short Sequence Insertion. Some methods [116], [259] involve inserting short sequences into long sequences to minimize padding. The Best-fit Packing [116] first splits long documents according to the model's context length, then sorts all document blocks in descending order of length. For each document block, it selects the training sequence set with the smallest remaining capacity that can accommodate it. [259] prioritizes long documents and uses a greedy algorithm to fill remaining space with short document segments (sequences), reducing padding and minimizing document concatenation to lower contextual noise.

Principles

Compared to traditional machine learning, LLMs place higher demands on the semantic quality of training data. Additionally, due to the requirement for uniform input lengths, a key challenge is maintaining semantic integrity without excessive truncation. Existing techniques tackle this through short-sequence insertion, sequence concatenation, and semantic-aware composition. However, it remains crucial to account for the impact of these data packaging operations on overall training efficiency.

Sequence Combination Optimization. Some methods [218], [316] optimize sequence combinations for efficient packing. [218] proposes two efficient sequence packing algorithms: (1) The Shortest Pack First Histogram Packing (SPFHP) uses a sequence length histogram, sorts sequences from long to short, and applies a worst-fit algorithm to prioritize placing the histogram intervals into the remaining largest "packs", while limiting packing depth to avoid creating excessive small packs, thus improving space utilization. (2) The Non-Negative Least Squares Histogram Packing (NNLSHP) converts the packing problem into a non-negative least squares problem, using dynamic programming to enumerate reasonable sequence combination strategies, constructing a packing matrix to determine the strategy's repetition count. It also assigns small weights to short sequences' residuals to reduce long sequence leftovers, achieving efficient packing. [316] splits documents into multiple fixed-length "buckets"

based on their length, ensuring that each sequence comes from the same document to avoid cross-document attention issues. Additionally, by combining Variable Sequence Length Curriculum (VSL), different lengths of sequences are dynamically sampled during training to maintain a consistent total token count.

Semantic-Based Packing. Some methods [364], [349] improve data coherence through semantic-based data packing. [349] reorders pretraining data by combining semantically related documents into coherent input contexts, allowing the LLM to read and reason across document boundaries. Similarly, SPLICE [364] randomly selects a document as the root document, and in a breadth-first manner, uses retrieval methods like BM25 and Contriever (trained from a mix of Wiki and CCNet data) to retrieve k similar documents, adding them to the training sample until the maximum length is reached. Finally, the tree structure is flattened using a specific tree traversal strategy to generate the training example.

2.5.4 Data Provenance

Data Provenance is the process of tracking the sources, transformations, and lineage of data, which is increasingly recognized critical in ensuring the reliability, transparency, and accountability of LLM data [54].

Principles

Compared with traditional machine-learning models, LLMs demand heightened safeguards for output security owing to their powerful generative capabilities. The central challenge is to preserve output integrity without degrading quality. Current solutions embed watermarks or deploy statistical-detection techniques to reveal any tampering.

Embedding Markers. Current data provenance methods [482], [105], [256], [212] generally modify the generation logic to embed covert markers into the text. This is done in a way that does not disrupt the text itself, thereby providing a medium for tracing the origin of the data.

Bileve [482] enhances the traceability and integrity of text by embedding two distinct levels of signals: ***(1) Statistical signal*** embedded globally to detect whether the text originates from a specific model. ***(2) Content-related signature*** embedded within each generation unit to verify if the text has been tampered with. During detection, the validity of the signature is first verified; if the signature is invalid, a statistical test is then used to determine whether the text comes from the target model.

Unlike Bileve that emphasizes strict traceability after text tampering, [105] focuses on embedding watermarks in a way that preserves the quality of the generated output. It embeds hidden markers that can only be detected by individuals possessing a specific key, while remaining imperceptible to others that the text has been altered. Specifically, the method employs a pseudo-random function (PRF, used to generate seemingly random numbers) to determine the shuffling of each output word, ensuring that the generated text is statistically indistinguishable from the original model's output. During detection, the presence of hidden markers is ascertained by calculating a score for each word in the text (based on the numbers generated by the pseudo-random function).

Unlike previous approaches, UPV [256] introduces a watermarking method that enables detection without requiring access to the key used during generation, thereby eliminating the risk of key leakage. It employs two independent neural networks for watermarking. During text generation, the watermark generation network utilizes an embedding module and a fully connected classifier to predict watermark signals based on token information within a sliding window, and accordingly adjusts the language model's output distribution. For detection, an LSTM-based network takes the text sequence as input and identifies the watermark, leveraging shared token embedding parameters with the generation network.

Compared to methods that require specific keys for detection, [131] embeds a special type of watermark into text generated by language models, which can be detected by anyone without the need for any secret information. It selects specific lexical combinations (rejection sampling, ensuring that the embedding of the marker does not affect the naturalness of the text) during text generation, in conjunction with an error correction mechanism (error-correcting codes, allowing the marker to be recovered even after partial modification of the text), to embed an encrypted signature (public key signature, ensuring the non-forgeability of the marker) into the text. During detection, one only needs to extract these specific *lexical combinations* from the text and verify the validity of the signature to determine whether the text contains the marker.

Statistical Provenance. Unlike the aforementioned methods that rely on detecting special markers for tracing the origin, [212] achieve data provenance through the statistical information of the vocabulary. Specifically, before generating each word, the model randomly divides the vocabulary into two parts (green-listed and red-listed tokens) and tends to favor the shuffling of green-listed tokens during the generation process (green-listed tokens are a randomly selected subset of the vocabulary). By employing statistical tests (a mathematical method used to determine whether text adheres to specific rules), it is possible to detect whether the proportion of green-listed tokens in the text is abnormal, thereby ascertaining if the text is machine-generated.

3 LLM for Data Management

After preparing the LLMs with carefully processed / stored / served data, we next introduce the LLM techniques that can be utilized to enhance data management tasks, including data manipulation, data analysis, and data system optimization.

3.1 LLM for Data Manipulation

LLM can be employed to explore and prepare appropriate data for non-LLM-oriented tasks, such as data cleaning for classification tasks, data integration for extracting well-structured tables from unstructured sources, and data discovery for identifying relevant datasets. Unlike data preparation pipelines designed specifically for LLM applications, these methods focus on enhancing the quality and utility of data for downstream analytical or machine learning tasks.

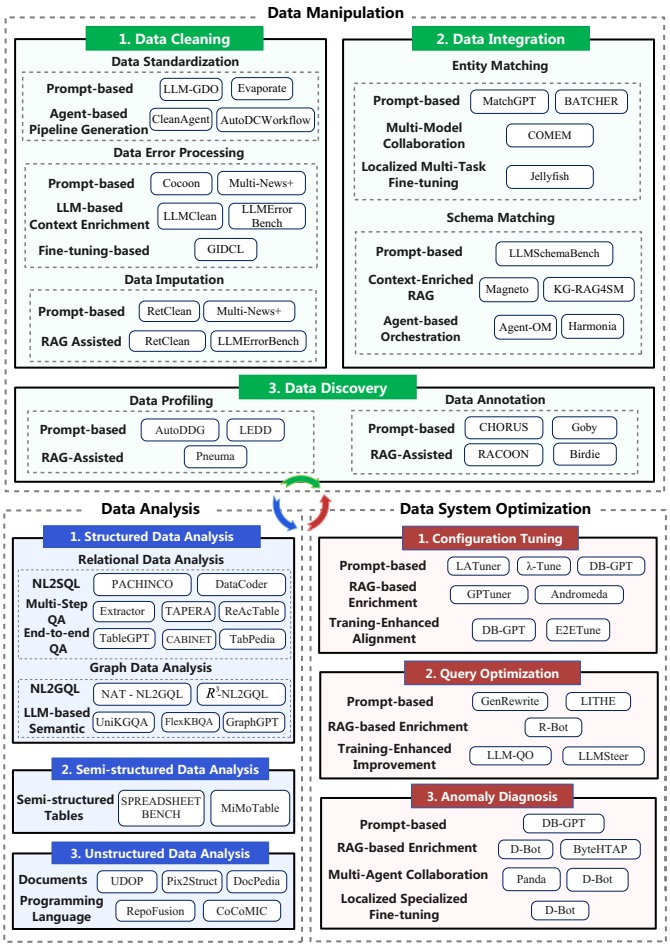

Fig. 8: Overview of LLM4DATA Techniques.

3.1.1 LLM *for Data Cleaning*

Data cleaning focuses on transforming corrupted or low-quality data into a reliable form suitable for downstream applications (e.g., statistical analysis or training machine learning models). It encompasses a range of tasks such as handling missing values, correcting typos, resolving formatting inconsistencies, and addressing dependency violations. These tasks are typically categorized into data standardization, error detection and correction, and data imputation.

Traditional data cleaning methods depend on rigid rules and constraints (e.g., zip code validation), demanding substantial manual effort and domain expertise (e.g., schema knowledge in financial data) [237], [432]. Additionally, they often require domain-specific training, which restricts their generalizability [63]. Recent studies show that large language models (LLMs) can address these limitations by offering natural language interfaces that reduce manual and programming effort, eliminate the need for complex runtime environments, and support seamless integration of domain knowledge. These methods primarily target the following tasks.

Data Standardization. Data standardization involves converting diverse, inconsistent, or non-conforming values into a consistent format to ensure reliable analysis and effective downstream processing. Existing methods use either structured LLM prompting for specific cleaning operations or LLM agents for automated pipeline generation.

(1) Prompt Based End-to-End Standardization. The first approach constructs well-structured prompts with explicit standardization instructions and employs advanced prompting techniques (e.g., Chain-of-Thought) to improve the effectiveness of LLM-based standardization methods. For example, LLM-GDO [279] utilizes user-defined prompts (UDPs), including in-context learning examples, to implement LLM-based operators that replace traditional user-defined functions (UDFs) across various standardization tasks (e.g., normalizing numerical values). This method simplifies logic implementation and facilitates the seamless integration of domain knowledge. Evaporate [63] employs LLMs to transform semi-structured documents into structured views through two main strategies: (i) *Evaporate-Direct*, which prompts the LLM to extract values directly, and (ii) *Evaporate-Code*, which guides the LLM to synthesize extraction code and ensembles multiple candidate functions using weak supervision to improve output quality while maintaining low cost.

(2) Agent Based Operation and Pipeline Generation. To address the inefficiencies of LLM-based solutions, such as the reliance on multi-turn prompts and expert-level prompt engineering, the second method employs LLM agents to automatically generate cleaning operations and orchestrate end-to-end pipelines. For instance, CleanAgent [319] integrates domain-specific APIs with autonomous agents to execute a standardization pipeline that includes API call generation (e.g., `clean_date(df, ''Admission Date'', ''MM/DD/YYYY'')`) and iterative code execution. Similarly, AutoDCWorkflow [237] adopts LLM agents to construct pipelines for resolving duplicates and inconsistent formats. The agent performs step-by-step reasoning to identify relevant columns, evaluate data quality, and generate appropriate operations (e.g., `upper()` and `trim()`), while leveraging tools such as OpenRefine for execution and feedback.

Data Error Processing. Given a data entry, error processing typically involves two steps: detecting erroneous values and correcting these values. Typical errors include typos, invalid formats, type mismatches, numeric outliers, and dependency violations. Existing methods generally fall into two categories: employing LLMs for direct end-to-end error processing, or enhancing context models to better guide the detection and correction process.

(1) Prompt Based End-to-End Error Processing. To support end-to-end data error processing, the first approach employs prompting techniques to either directly handle data errors or generate the corresponding processing functions. For instance, Multi-News$^+$ [103] employs Chain-of-Thought (CoT) prompting, majority voting inspired by human annotation practices, and self-consistency checks to enhance classification accuracy and transparency when processing noisy documents. Similarly, Cocoon [461] constructs semantic detection prompts and divides datasets into batches, allowing the LLM to analyze sampled values (e.g., 1,000 entries per column) and identify typos or inconsistencies (e.g., "mapping English" $\rightarrow$ "eng"), thereby supporting batch-wise data cleaning. GIDCL [432] adopts a creator-critic framework in which the LLM iteratively refines lightweight error detection models and generates pseudo-labeled data using handcrafted prompts and in-context examples to produce both detection and correction functions, further enhanced by structural correlation learning with Graph Neural Networks (GNNs).

(2) LLM Based Cleaning Context Enrichment. To address the inefficiencies and limited scalability of manual clean-

ing context model construction in dynamic environments, the second approach leverages LLMs to enrich data cleaning context models and more effectively capture semantic relationships within the data. For example, LLMClean [78] proposes an automated LLM-based method for generating context models by extracting ontological functional dependencies (OFDs) using both prompt ensembling and fine-tuned LLMs (e.g., Llama-2). The extracted OFDs are then used to identify data errors (e.g., value inconsistencies) and guide LLM-based repairs through iterative feedback from integrated correction tools such as Baran. LLMErrorBench [74] employs LLM agents equipped with Python (via IPython) and prompted with task-specific instructions and contextual hints (e.g., error locations) to explore, modify, and repair datasets iteratively. Corrections (e.g., value replacement, missing data handling) are guided by performance feedback from pre-defined code execution and evaluation pipelines.

(3) Fine-tuning Based End-to-End Error Processing. To improve error correction accuracy while preserving computational efficiency and model adaptability, the third approach fine-tunes LLMs to capture dataset-specific patterns and dependencies that are typically difficult to model through prompting alone. For example, GIDCL [432] fine-tunes a local LLM (e.g., Mistral-7B) using Low-Rank Adaptation (LoRA) to optimize error correction, constructing training data from labeled tuples and pseudo-labeled tuples generated via LLM-based augmentation, with each training instance formatted as a context-enriched prompt comprising: (i) an instruction (e.g., "Correct the ProviderID to a valid numeric format"), (ii) a serialized erroneous cell with row and column context (e.g., "`<COL>ProviderID<VAL>1x1303...`"), (iii) in-context learning demonstrations (e.g., "bxrmxngham → birmingham"), and (iv) retrieval-augmented examples from the same cluster (e.g., clean tuples via k-means).

Data Imputation. Given a data entry with missing attribute values (e.g., NULL), data imputation aims to infer the missing values using available contextual information accurately. Existing methods either (i) use structured prompts to convey contextual hints to LLM, or (ii) apply retrieval-augmented generation (RAG) to integrate relevant external data.

(1) Prompt Based End-to-End Imputation. To incorporate contextual information for imputing missing values, the first approach constructs structured prompts. For example, RetClean [129] enhances LLM effectiveness by serializing each tuple into a formatted representation (e.g., "[Name: John; Age: 25; Gender: NULL]") and pairing it with a targeted question such as "What is the correct value for Gender?". This prompt design enables the LLM to generate accurate, context-aware missing values.

(2) RAG Assisted Localized Imputation. To enable online LLMs in handling unseen, domain-specific, or private datasets, the second approach adopts the retrieval-augmented generation (RAG) paradigm. For example, RetClean [129] introduces a retrieval-based data cleaning framework that indexes a data lake using both syntactic (Elasticsearch) and semantic (Faiss/Qdrant) methods. It retrieves the top-k relevant tuples, reranks them (e.g., using ColBERT), and then leverages an LLM to infer missing values, while maintaining lineage tracking for transparency and traceability.

3.1.2 LLM *for Data Integration*

Data integration aims to align elements across heterogeneous datasets to enable unified access, analysis, and knowledge extraction. For instance, it includes identifying tables or records that correspond to the same real-world entity. Moreover, it facilitates downstream tasks such as data augmentation by establishing semantic relationships across sources.

Traditional integration methods often struggle with semantic ambiguities and conflicts, particularly in complex integration scenarios without domain-specific knowledge [277]. Furthermore, classical models (e.g., pretrained models) generally require large amounts of task-specific training data and tend to degrade in performance when encountering out-of-distribution entities [308]. In contrast, recent studies have shown that LLMs possess strong semantic understanding, enabling them to uncover correlations across datasets and incorporate domain-specific knowledge, thereby offering robust generalization across diverse integration tasks.

Entity Matching. The goal of entity matching is to determine whether two entries refer to the same real-world entity. Existing methods leverage LLMs through well-structured prompts and advanced reasoning mechanisms, incorporate multiple models for collaborative matching, and apply multi-task fine-tuning to further enhance performance.

(1) Prompt Based End-to-End Matching. To improve LLM's effectiveness on matching tasks, the first approach crafts well-structured prompts and integrates auxiliary mechanisms to strengthen the robustness of the reasoning process.

• *Manually-Crafted Prompt.* This method incorporates detailed instructions and illustrative examples into the prompts to guide LLM in performing entity matching more effectively. For example, MatchGPT [308] evaluates the performance of both open-source and closed-source LLMs (e.g., Llama 3.1 and GPT-4o mini) with (i) different prompt designs, (ii) the selection of in-context demonstrations, (iii) automatic generation of matching rules, and (iv) fine-tuning LLMs using a shared pool of training data. To reduce inference costs, BATCHER [134] introduces a batch prompting method that allows multiple entity pairs to be processed simultaneously. It optimizes in-context learning by (i) grouping entity pairs into a single prompt and (ii) applying a greedy cover-based strategy to select demonstrations such that each query in the batch is semantically close to at least one example.

• *Pseudo-Code Guided Reasoning.* To mitigate hallucinations arising from over-reliance on an LLM's internal knowledge, this method integrates external formalized representations to enhance the robustness and reliability of the reasoning process. For example, KcMF [430] guides LLMs using expert-designed pseudo-code instructions structured as a sequence of if-then-else logical conditions, combined with external domain knowledge (e.g., datasets and examples). It further adopts an ensemble strategy by generating outputs from different knowledge sources (e.g., Wikidata and domain-specific datasets) and applies a voting mechanism to aggregate results, improving consistency and accuracy.

(2) End-to-End Matching with Multi-Model Collaboration. To leverage the strengths of different models across tasks, the second approach employs collaborative entity matching using models of varying sizes. For example, COMEM [400] introduces a compound entity matching framework that combines multiple strategies with LLM collabo-

ration to address global consistency, which is often ignored in binary matching. It employs (i) a local strategy using a medium-sized LLM (3B-11B) as a matcher or comparator to rank top-k candidates via bubble sort, reducing position bias and context length dependency; and (ii) a global selection strategy using a stronger LLM (e.g., GPT-4o) to refine top-k candidates by modeling inter-record interactions.

(3) Localized* LLM *Fine-tuning of Multi-Task Learning. To enhance the generalization capability of local LLMs, the last approach integrates multiple task-specific datasets within a unified multi-task instruction tuning framework. For example, Jellyfish [454] applies parameter-efficient instruction tuning to locally deployed LLMs (7B-13B) across diverse data processing tasks. It employs techniques such as chain-of-thought prompting over task-specific serialized data and reasoning data distillation, using explanation traces generated by a larger mixture-of-experts model (Mixtral-8x7B-Instruct) to guide the learning process.

Schema Matching. The objective of schema matching is to identify correspondences between elements of different database schemas (e.g., matching attribute names "employee ID" and "staff number"). Existing approaches directly apply prompting techniques to enable LLMs to perform end-to-end matching, utilize retrieval-augmented generation (RAG) to enhance contextual understanding, and employ LLM agents to orchestrate the overall matching workflow.

(1) Prompt Based End-to-End Matching. To facilitate schema matching without requiring rigid code implementations, the first method employs various prompting techniques to guide LLM in identifying the desired mappings. For example, LLMSchemaBench [304] applies prompt engineering techniques to interact with LLMs, defining four task scopes that differ in the level of contextual information included in the prompts. The prompts are constructed using established design patterns: the persona pattern (e.g., instructing the LLM to act as a schema matcher), meta language creation (e.g., explicitly defining valid match criteria), Chain-of-Thought reasoning, and the output automater (e.g., generating structured JSON outputs for downstream automation).

(2) End-to-End Matching via Context-Enriched RAG. To enrich the matching context and improve accuracy, the second method integrates retrieval-augmented generation (RAG) with various strategies. For example, Magneto [267] employs a retrieve-rerank framework that combines small pre-trained language models (SLMs) with LLMs to deliver cost-effective and generalizable schema matching. SLMs serve as candidate retrievers, generating an initial ranked list of potential matches from the target table for each input column, which is then refined by LLMs acting as rerankers to improve accuracy. KG-RAG4SM [277] incorporates multiple retrieval strategies, including vector-based, graph traversal-based, and query-based, to extract relevant subgraphs from knowledge graphs (KGs). These subgraphs are further refined through ranking mechanisms and used to augment LLM prompts, thereby improving schema matching performance through enriched contextual input.

(3) Agent-Based Matching Workflow Orchestration. To address complex matching patterns, the final approach leverages LLM-based agents to orchestrate the end-to-end matching workflow. For example, Agent-OM [320] employs two LLM agents (i.e., Retrieval Agent and Matching Agent) to control the workflow by decomposing tasks via Chain-of-Thought (CoT) prompting, invoking specialized tools (e.g., syntactic/lexical/semantic retrievers and matchers), and accessing a hybrid database (relational + vector) for memory storage and retrieval. Harmonia [340] leverages LLM-based agents to orchestrate data harmonization tasks, combining predefined data integration primitives (e.g., schema matching, value matching) with on-demand code generation when the primitives are insufficient. In addition, it employs techniques like ReAct for reasoning and action planning, interactive user feedback for error correction, and declarative pipeline specifications for reproducibility.

3.1.3 LLM *for Data Discovery*

Data discovery focuses on identifying relationships within datasets through tasks like data annotation (e.g., column type classification) and profiling (e.g., metadata generation). Unlike data analysis, which emphasizes statistical computations or factual answer generation, data discovery enables deeper semantic understanding critical for downstream applications such as integration, search, and recommendation.

Existing data discovery methods face two limitations. First, they typically consider limited interaction between queries and tables [163]. Second, many of these approaches rely heavily on large training datasets, struggle with distribution shifts, and fail to generalize to rare or domain-specific data [143], [217]. Recent studies have shown that LLMs can effectively address these challenges by generating high-quality metadata, enriching dataset context, and supporting natural language interfaces for data discovery tasks.

Data Profiling. Data profiling typically involves characterizing a given dataset by generating additional information (e.g., dataset descriptions). Recent methods often employ prompting techniques to guide LLM in generating such metadata by leveraging their pretrained knowledge and contextual understanding.

(1) Manually Crafted Profiling Prompt Engineering. To profile different aspects of a dataset without extensive manual effort or code implementation, the first approach relies on a set of manually crafted profiling prompts. For example, AutoDDG [456] utilizes LLM with carefully designed prompts to generate two types of descriptions (i.e., User-Focused Descriptions (UFDs) for readability and Search-Focused Descriptions (SFDs) for search optimization) tailored to the dataset's content and intended usage. LEDD [58] employs carefully crafted prompts to support core data discovery tasks in data lakes. For hierarchical cataloging, prompts instruct LLM to summarize data clusters into semantically meaningful categories. For semantic search, prompts refine natural language queries before embedding and retrieval. For real-time relation analysis, prompts guide LLM in comparing expanded graph nodes and describing inter-table relationships.

(2) RAG Assisted Context Enrichment. To enhance retrieval effectiveness across diverse query types, the second method adopts a hybrid approach that integrates diverse retrieval techniques. For example, Pneuma [72] adopts a RAG framework to retrieve relevant tables from databases, data lakes, or repositories based on natural language queries. It combines LLMs with traditional retrieval techniques, such as full-text and vector search, using LLMs for both schema

narration (i.e., generating meaningful column descriptions) and as judges to refine and rerank retrieved results.

Data Annotation. Data annotation involves assigning semantic or structural labels to data elements, such as identifying column types (e.g., *Manufacturer* or *birthDate* from the DBPedia ontology). Recent methods leveraging LLM typically design prompts with task-specific annotation instructions. Additionally, some approaches employ retrieval-augmented generation (RAG) techniques and the contextual reasoning capabilities of LLMs to further enrich the annotation context and improve performance.

(1) Task-Specific Annotation Prompt Engineering. To flexibly support diverse annotation tasks, the first approach encodes task-specific instructions and requirements within carefully crafted prompt templates. For example, CHORUS [203] integrates LLMs into the annotation pipeline using task-specific prompts that incorporate instructions, demonstrations, data samples, metadata, domain knowledge, and output formatting guidance. Goby [204] explores the use of LLMs for semantic column type annotation in a domain-specific enterprise setting by crafting a set of tailored prompts. It proposes several techniques to improve performance, including tree serialization (providing the full ontology as prompt context), grammar-constrained decoding (enforcing hierarchical structure during generation), and step-by-step prompting (Chain-of-Thought strategy to guide ontology navigation). LLMCTA [217] evaluates diverse LLMs for generating and refining label definitions by employing methods like knowledge generation prompting (e.g., producing initial demonstrations), self-refinement (error-based definition improvement), and self-correction (two-step pipeline featuring a reviewer model).

(2) RAG Assisted Annotation Context Enrichment. To supply LLM with relevant annotation context, the second approach utilizes diverse retrieval strategies within retrieval-augmented generation (RAG) frameworks to enrich the input.

• *Classical Retrieval Technique.* To mitigate the shortcomings of vanilla LLM-based annotation, such as outdated knowledge, this method augments the context with retrieved external knowledge. For example, RACOON [408] performs semantic type annotation by leveraging a Knowledge Graph (KG) to retrieve entity-related information (e.g., labels and triples) associated with column cells. This information is then processed into concise contextual representations and incorporated into LLM prompts to improve annotation accuracy.

• *LLM Based Generation.* To fully leverage LLM's internal knowledge, this method relies on the model itself to generate relevant contextual information. For example, Birdie [163] leverages LLMs to automatically generate natural language queries for training a differentiable search index (DSI), which facilitates linking relational tables to queryable knowledge by enriching them with contextual semantics. It supports scalable structured data annotation, using prompts composed of structured markdown tables comprising captions, headers, and sample rows alongside explicit task instructions.

3.2 LLM for Data Analysis

Apart from data manipulation, LLMs hold the potential to revolutionize traditional data analysis paradigms by supporting natural language interfaces and enabling advanced, semantic-aware analysis tasks that typically require human involvement. In this section, we discuss the challenges and techniques of LLM-based data analysis, including structured data analysis, semi-structured data analysis, and unstructured data analysis.

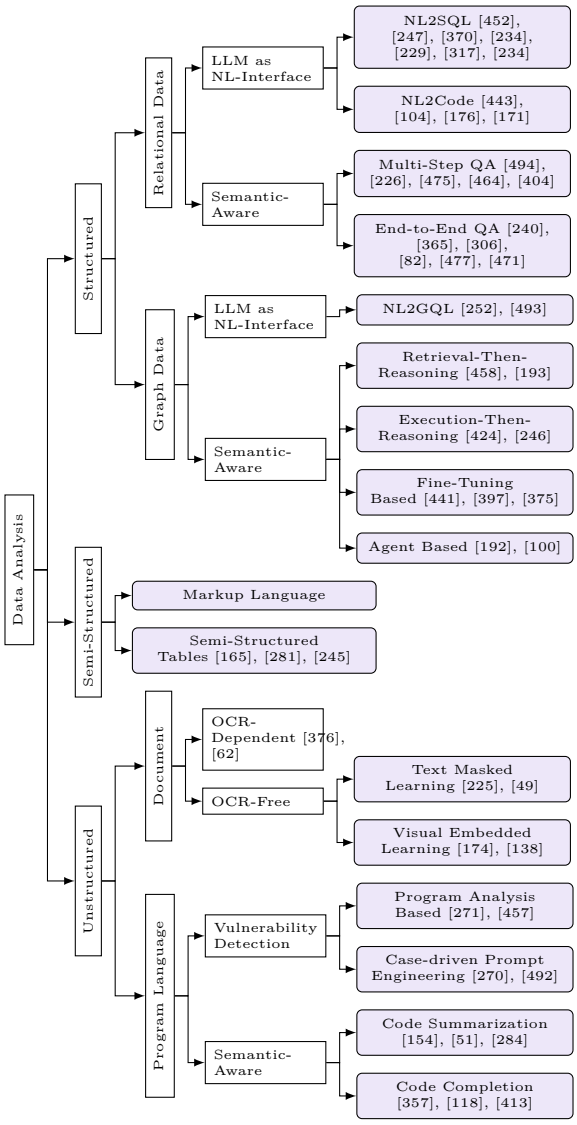

Fig. 9: Overview of LLM for Data Analysis.

3.2.1 LLM *for Structured Data Analysis*

Structured data refers to data with well-defined schemas like relational (tabular) data [107] and graph data [60].

3.2.1.1 *Relational Data Analysis*

LLM for Natural Language Interfaces. Basic analysis jobs for relational data are typically characterized by well-defined operations. These include basic calculations (e.g., summation, averaging, counting, ranking), statistical analysis (e.g., regression, K-means clustering), and data quality assurance processes (e.g., constraint validation, outlier detection). Such tasks can generally be supported by tools like SQL or Python libraries (e.g., Pandas).

(1) NL2SQL. With the help of LLM, users can directly perform operations using natural language. NL2SQL focuses on

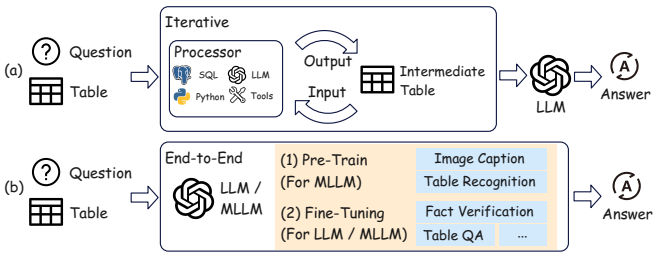

Fig. 10: **General Workflows** - (a) Multi-Step Relational Data QA. (b) End-to-End Relational Data QA.

translating natural language queries into SQL commands by leveraging techniques such as (*i*) schema linking, which aligns user intents with database schema to resolve ambiguities [452], [247], (*ii*) content retrieval, which dynamically extracts relevant information from the database to refine query generation [370], [234], and (*iii*) SQL generation strategies such as multi-step generation, intermediate SQL representation, and different decoding strategies [229], [317], [234], [483], [484].

(2) NL2Code. Different from NL2SQL, NL2Code approaches emphasize enhancing relational data analysis through generating Python code (e.g., Pandas, NumPy), which includes a vast number of library APIs characterized by high variability and complexity, and often requiring the handling of complex chain operations. Recent advancements address these issues to some extent.

• *Model Finetuning:* PACHINCO [443] fine-tunes a 62B parameter PALM [104] model in two stages (i.e., separately using a Python source code corpus with 64B tokens and a Jupyter notebook corpus with 9.6B tokens) so as to improve model performance on analysis-related tasks (e.g., calculate the amount of games added in each year for each month). DataCoder [176] utilizes different types of contexts (e.g., code, text, and data) by employing dual encoders (e.g., data encoder and code + text encoder) and one general decoder to generate code in notebooks.

• *LLM Based Analysis Agent:* Data Interpreter [171], on the other hand, leverages LLMs through APIs to generate task and action graphs. Specifically, they utilize LLM's semantic reasoning ability to accurately decompose complex user queries into subproblems (e.g., correlation analysis, data exploration, and anomaly detection), and refine and verify each subproblem to improve code generation results for data science tasks.

LLM for Semantic Analysis. Moreover, some jobs require LLM-based analysis, such as those that involve semantic understanding or demand outputs in natural language format (e.g., table summarization). These challenges call for methodologies like (1) multi-step question answering (QA) with diverse decomposition strategies and (2) end-to-end QA leveraging specifically optimized LLMs.

• ***Multi-Step QA.*** Multi-step question answering (QA) refers to decomposing complex queries into a sequence of subquestions to facilitate step-by-step reasoning. According to the question decomposition mechanisms, existing methods can be categorized into two types: (1) static decomposition, which follows predefined and fixed processing steps (e.g., retrieve-select-reason), and (2) LLM-driven iterative decomposition, in which the LLM dynamically determines the next operation based on the contextual history of the reasoning process.

(1) Static Decomposition. The static decomposition includes Retriever-Selector-Reasoner frameworks and the variants, which partition tasks into modular components for better multi-step inference and enhanced interpretability. The Extractor-Reasoner-Executor paradigm [494] extracts the relevant segments from the context, generates the logic rules or equations, and performs the rules or executes the equations to get the final answer through LLM prompting. S3HQA [226] trains a retriever which aims to perform initial filtering of heterogeneous resources, utilizes a selector to select the most relevant factual knowledge, and a generation-based reasoner to obtain final answers.

(2) Iterative Decomposition. However, static decomposition paradigm performs poorly on multi-hop queries, while LLM-driven iterative decomposition, which dynamically refines subtasks through recursive reasoning, could effectively address the issue.

TAPERA [475] introduces the query decomposition step into the question answering process by adopting the LLM-driven approach. The Planner decomposes the query into subqueries, forming an initial plan. The Reasoner then generates executable programs for each sub-query, while the Answer Generator derives answers based on the program outputs to fulfill the plan. Finally, the Planner updates or finalizes the plan as needed.

Similarly, ReAcTable [464] and CHAIN-OF-TABLE [404] iteratively generate operations and update the table to present a reasoning chain as a proxy for intermediate thoughts through prompting LLMs and in-context learning.

• ***End-to-End QA.*** End-to-End Question Answering (QA) refers to approaches in which the answer-generating LLM directly produces the final response without intermediate steps or iterative refinement. Based on the data representation and processing mechanisms, the relevant methods can be classified into table-specific LLM fine-tuning, table content retrieval, and table-as-image analysis.

(1) Table-Specific LLM Fine-Tuning. Fine-tuning LLMs on task-specific table datasets enables them to internalize analytical knowledge directly within their parameters. TableGPT [240] fine-tunes LLMs like GPT-3.5 using a diverse set of table tasks synthesized from real-world tables. Building on Qwen2.5 [324], TableGPT2 [365] introduces a table encoder to generate a hybrid table representation, an adapter to generate query representations, and a LLM decoder generates an agent workflow (i.e., the tool execution pipeline) to derive the final answer. The TableGPT2 model is pre-trained on 593.8K tables and fine-tuned 2.36M question-answer pairs.

(2) Table Content Retrieval. Instead of embedding the whole table, table content retrieval enhances model performance by eliminating noisy parts of the table while retaining information relevant to question answering. CABINET [306] employs a weakly supervised component to produce a parsing statement that defines the criteria for selecting relevant rows and columns, emphasizing the corresponding table cell content. TableMaster [82] constructs a refined subtable through row and column lookup. By leveraging carefully designed LLM prompts (e.g., provide objective, table definition, table information, question, instructions, and response format), it

ranks all candidate columns, selects a relevant subset based on the query, and then instructs the LLM to generate an SQL query for extracting the most relevant rows.

(3) Table-As-Image Analysis. Due to the limitations of (text-only) LLMs in understanding table structures, the Table-as-Image approach has been proposed, converting tables into images for analysis using multimodal LLMs. Table-LLaVA [477] applies incremental pretraining to LLaVA-7B [258] on 150K table recognition samples (e.g., input a table image and output table representations in HTML, Markdown, or LaTeX), enabling the model to align table structures and elements with textual modality. It is further fine-tuned on 232K samples on question answering, text generation, fact verification, and structure understanding tasks to enhance its instruction-following ability. To enable a single model to perform various analytical tasks, TabPedia [471] introduces the concept synergy mechanism, abstracting all table analysis tasks into concepts. Built on Vicuna-7B [476], it appends meditative tokens to the input of the LLM decoder, which adaptively activates different regions of visual tokens and helps the model interpret the intent behind specific task questions. However, such methods face limitations when processing twisted or distorted tables, and their performance degrades significantly when directly handling document images.

3.2.1.2 Graph Data Analysis

Different from relational data, graph data represents entities (vertices) and their inter-dependencies (relationships) to explicit model of complex network semantics (e.g., social networks and knowledge graphs) beyond rigid tabular schema, which presents unique challenges due to the vast search space and complex path reasoning in multi-hop queries [59]. Compared with relational data analysis, graph data analysis involves more complex jobs like summarization based on the multi-hop relations across the graph vertices and reasoning over text-attributed graphs whose nodes and edges are associated with text [252], [493]. Graph data can not only be stored in relational databases, but also be stored and queried in knowledge graphs and accessed through SPARQL in RDF databases (e.g., Blazegraph [8] and GraphDB [21]) or Cypher in Neo4j [17].

Traditional graph analysis (e.g., statistical methods, graph neural network (GNN) based methods) encompasses a spectrum of tasks, including *node classification* (e.g., categorizing academic papers into research domains), *graph classification* (e.g., predicting node properties over molecular graphs), *link prediction* (i.e., inferring latent relationships between graph nodes), *community detection* (i.e., identifying densely connected subgraphs), *anomaly detection* (i.e., identifying deviations from expected patterns), *graph clustering*, and etc. However, these methods have their own limitations. Statistics-based methods fail to handle complex semantic information (e.g., query can be extremely complex and requires human expertise), while graph neural networks (GNNs) exhibit limited generalization capabilities, necessitating task-specific retraining on different tasks.

In contrast, the advent of LLMs offers transformative potential by leveraging their advanced reasoning capacities and cross-domain generalization abilities, which can (1) simplify the query writing costs (e.g., NL interfaces) and (2) achieve semantic-aware analysis unsupported in traditional ones.

Natural Language To Graph Analysis Query. Different from NL2SQL, the syntax of graph query language generation is more complex (i.e., MATCH, LOOKUP, GET and other operations unique to graph data manipulation) and there exist two operation objects (i.e., vertex and edge) [493]. By integrating natural language interfaces with graph data, LLMs facilitate flexible and efficient query generation without the need for specialized model architectures.

To enhance LLMs' comprehension of the complex syntax of Graph Query Language (GQL), R^3-NL2GQL [493] proposes a hybrid approach leveraging relatively small LLM (e.g., LLaMA3-7B) as a selector and GQL rewriter, while employing a larger LLM (e.g., GPT-4) as a reasoner. The selector identifies the necessary CRUD functions, clauses, and schema, while the rewriter refines the query by aligning it with the relevant graph data retrieved by minimum edit distance and semantic similarity calculation. The LLM then synthesizes the aligned question, selected operations, and schema to generate the final GQL query.

To address the limitations of LLMs in planning and collaborating with other LLMs, NAT-NL2GQL [252] introduces a three-agent framework. The Preprocessor agent constructs context information, including query rewriting, path linking, and the extraction of query-relevant schemas. The Generator agent, an LLM fine-tuned with NL-GQL data, generates GQL statements based on the rewritten queries and extracted schemas. The Refiner agent iteratively enhances the GQL or contextual information by leveraging error feedback from GQL execution results.

Note that, within the context of AI for Science (AI4Science), the integration of LLMs with graph data analysis has also shown significant potential and wide-ranging applications (e.g., treat polymers as graphs and predict their properties [242], [309]), which is not the primary focus of this survey.

LLM-based Semantic Analysis. Furthermore, certain jobs necessitate semantic-aware analysis, such as summarizing textual paragraphs embedded within graph nodes. Based on the adopted LLM strategies, we classify the relevant methods into retrieval-then-reasoning methods, execution-then-reasoning methods, graph task based fine-tuning methods, and agent based methods.

- ***Retrieval-Then-Reasoning.*** Retrieval-then-reasoning first extracts a question-specific subgraph from the graph to identify the most relevant entities and then generates answers using LLMs. To address the challenge of a vast search space, [458] introduces a two-stage approach. First, a trainable and decoupled subgraph retriever selects a relevant subgraph based on the query. Then, reasoning is performed over the retrieved subgraph to derive the final answer. UniKGQA [193] integrates retrieval and reasoning within a unified model architecture. It comprises a semantic matching module, leveraging a pre-trained RoBERTa [266] for the semantic alignment between questions and relations in graphs, and a matching information propagation module that propagates matching signals along directed edges in graphs.
- ***Execution-Then-Reasoning.*** Execution-then-reasoning refers to the process of parsing natural language queries into executable logical forms (e.g., SPARQL) that align with the graph data, followed by reasoning based on the output of the executed program. Interactive-KBQA [424] introduces an in-

teractive LLM QA framework with a unified SPARQL-based toolset (e.g., entity search, graph pattern search, SPARQL execution, etc.) designed to address complex queries. FlexKBQA [246] addresses the challenge of lacking high-quality annotated data in real-world scenarios. By prompting LLMs as program translators, it samples program-answer pairs from the knowledge base and generates corresponding natural language questions. The synthetic question-program-answer dataset is used to train lightweight models through execution-guided self-training, which are subsequently employed to annotate real user queries. This approach addresses the distribution shifts between synthetic and actual data, leading to significant improvements in few-shot learning scenarios.

• ***Graph Task Based Fine-tuning Methods.*** InstructGLM [441] enables generative graph learning by fine-tuning an LLM and leveraging natural language descriptions of graph structures (e.g., offer the first node and the 1-/2-/3-hop neighbors' information). InstructGraph [397] introduces a stricter code-like graph representation format which constructs entities and triples in the form of list, whose backbone LLM (LLaMA2-7B) is fine-tuned on a graph-centric corpus comprising 1.6 million instances. To mitigate the issue of hallucination, it incorporates Direct Preference Optimization (DPO) algorithm [329] for preference alignment. GraphGPT [375] enhances model performance in zero-shot scenarios by incorporating a structural information encoding module based on Graph-SAGE [166] and GCN [211]. It fine-tunes the projector bridging the graph encoder and the LLM decoder to align the language capabilities of the foundation LLM (Vicuna-7B) with the graph learning tasks.

• ***Agent Based Methods.*** Agent-based methods involve leveraging LLM-based agents with predefined tools (e.g., human-written interfaces or graph processing library APIs) that iteratively interact with the graph data to retrieve, refine, and operate information. StructGPT [192] introduces an iterative reading-then-reasoning framework, leveraging specialized interfaces to operate on graph data. It repeatedly applies an invoke-linearize-generate procedure to derive query results. Another approach is to generate an entire reasoning path based on the query and refine it only when necessary. Readi [100] initially constructs a reasoning path and instantiates it on the graph. When execution errors occur, it collects error messages and invokes an LLM to revise the path. The final answer is inferred from the instantiated graphs.

3.2.2 LLM *for Semi-Structured Data Analysis*

Semi-structured data refers to data that are neither with strictly predefined schema like relational models nor raw data (e.g., plain text or images) [48]. Meanwhile, they still maintain part of organizational properties (e.g., tags, headers) and have hierarchical or nested representation (e.g., *County* - *Province* - *City* in a nested JSON).

3.2.2.1 Markup Language

Markup languages (e.g., XML, JSON, and HTML) are widely used for structuring and exchanging data across systems. Traditional approaches for processing these formats typically involve transforming them into structured tables or representing them as hierarchical tree structures. Leveraging the reasoning capabilities of LLMs, it becomes possible to directly extract and interpret hierarchical relationships, attributes, and nested structures from data without the need for intermediate transformations.

3.2.2.2 Semi-Structured Tables

Compared to structured relational data, semi-structured tables exhibit a more complex structural organization characterized by merged cells. This inherent complexity presents a significant challenge in aligning queries with the table content and structure in query answering tasks. The lack of efficient tools (usually using the openpyxl library) and representation methods (usually stored in Excel or HTML files) for handling semi-structured tables makes it more difficult to process such data.

Although research on semi-structured table analysis is limited, several studies have compiled various semi-structured table reasoning datasets, providing valuable data support. TEMPTABQA [165] consists of 11,454 question-answer pairs focused on temporal queries, while SPREADSHEETBENCH [281] presents a challenging benchmark for spreadsheet manipulation, with 912 questions derived from real-world scenarios. MiMoTable [245] incorporates reasoning across multiple sheets and files, containing 1,719 queries within 428 spreadsheets. Evaluation results on these benchmarks highlight a significant performance gap (ranging from 20% to 50%) between state-of-the-art models and human performance, calling for further exploration in this area.

3.2.3 LLM *for Unstructured Data Analysis*

Unstructured data refers to data that lacks explicit structure, as it does not adhere to a predefined schema. Additionally, it exhibits high variability in format, length, and modality, which further complicates its processing and analysis.

3.2.3.1 Documents

Documents exhibit complex layouts and styles with diverse elements, including a hybrid of images, tables, charts, plain text, and formulas.

• ***OCR-Dependent Methods.*** OCR-based methods refer to approaches that involve performing Optical Character Recognition on document images, followed by the integration of textual, layout, and visual features for reasoning. UDOP [376] integrates text and layout modalities within a unified encoder, dynamically fusing image patch tokens and text tokens based on their spatial information. Specifically, when the center of a text token's bounding box falls within an image patch, the corresponding image patch embedding is added to the text token embedding, enabling a more cohesive representation of document structure. DocFormerV2 [62] preserves the integrity of layout information by employing a visual encoder. Image patches and text bounding box positions are embedded through a linear layer and added to the corresponding token embeddings as input to the T5 [331] encoder. To achieve local feature semantic alignment, the model undergoes pretraining on token-to-line (i.e., predict whether a key-value pair is on the same line or adjacent lines) and token-to-grid (i.e., predict each token located in which image grid) tasks. The T5 decoder is then incorporated to fine-tune the whole model on downstream tasks.

• ***OCR Free Methods.*** However, the OCR step often introduces semantic errors, resulting in suboptimal performance. To fill this gap, OCR-free methods have emerged, directly

generating the target token sequences with end-to-end multimodal LLMs [257], [407]. Based on different approaches to enhancing model understanding of textual semantics, related works can be categorized into text masked learning and visual embedded learning.

(1) Text Masked Learning. Text Masked Learning involves masking textual content within a document and training the model to predict the missing text. Pix2Struct [225] is a typical vision-encoder-text-decoder pre-trained image-to-text model designed for visual language understanding based on ViT [124]. It is pretrained to parse masked web pages into simplified HTML. The model introduces a variable-resolution input representation, rescaling input images to maximize the number of patches that can fit within the given sequence length, to prevent aspect ratio distortion. DUBLIN [49] designed multiple fine-tuning tasks (i.e., bounding box prediction based on given text, text prediction based on given bounding box, masked text generation, and query answering) to improve the generalization ability.

(2) Visual Embedded Learning. In Visual Embedded Learning, there are no specially designed training objectives. Instead, the model is directly fine-tuned on downstream tasks to enhance its understanding of textual content within images. mPLUG-DocOwl1.5 [174] introduces a spatial-aware vision-to-text module designed for representing high-resolution, text-rich images. This module preserves structural information while reducing the length of visual features. It consists of a convolution layer to shorten the sequence length and a fully connected layer that projects visual features into the language embedding space. Unlike most methods that crop or resize the initial image before feeding it into a vision encoder, DocPedia [138] directly processes visual input in the frequency domain. It utilizes JPEG DCT [388] extraction to obtain DCT coefficients, which are then processed using a frequency adapter before being input into the vision encoder. This approach allows the model to capture more visual and textual information while using a limited number of tokens. The performance improvement observed in the experiment suggests that this method offers a novel approach for processing high-resolution images.

3.2.3.2 Program Language Analysis

Programming language analysis involves multiple levels of abstraction, including lexical analysis, parsing, and semantic analysis, each requiring distinct techniques to process source code effectively. Additionally, it must handle both local and global information, such as variable scopes, function call chains, and complex dependencies, which pose significant challenges for accurate program understanding.

LLM as Program Vulnerability Detection Tools. Recent advancements in LLMs have opened new avenues for improving vulnerability detection tools. Training LLMs based on program analysis techniques enhances their ability to understand programs at both the lexical and syntactic levels. Leveraging in-context learning through case-driven prompt engineering enhances the model's accuracy by providing relevant examples.

• ***Program Analysis based Training.*** Static and dynamic program analysis are commonly used methods for detecting vulnerabilities in programs. By assisting these processes, LLMs improve the accuracy of vulnerability detection. PDBER [271] is a model fine-tuned on CodeBERT [141] through three tasks (i.e., Predicting Masked Tokens, Predicting Statement-Level Control Dependencies, and Predicting Token-Level Data Dependencies). This enables more fine-grained vulnerability analysis at the statement level. To reduce the impact of irrelevant information, [457] decomposes the control flow graph (CFG) into multiple execution paths from the entry node to the exit node. CodeBERT and a CNN are employed to capture intra-path and inter-path representations, respectively. The extracted feature vectors are then combined as a unified program representation, which serves as input to a MLP classifier for vulnerability detection.

• ***Case-driven Prompt Engineering.*** Leveraging the in-context learning and few-shot learning capabilities of LLMs can significantly improve their accuracy in vulnerability detection. VUL-GPT [270] uses GPT-3.5 to generate analysis content (i.e., the program interpretation) for the input code and retrieves similar code snippets and corresponding vulnerability information through BM25 [338] or TF-IDF. The retrieved information, along with the original code and analysis, is then input into GPT to detect vulnerabilities. [492] designs various prompts, such as random code samples and retrieve-based code samples, and demonstrates that GPT-4 outperforms state-of-the-art models in vulnerability detection.

LLM-based Semantic-aware Analysis. Traditional semantic-aware tasks convert programs into ASTs [362] or graph structures [151] and train Seq2Seq models to learn program syntax, dependencies, and semantics. However, these approaches lack general knowledge, leading to limited generalization ability. By leveraging the world knowledge and few-shot learning capabilities of LLMs, the performance of tasks such as code summarization and code completion has been significantly improved.

• ***LLM as Code Summarizer.*** Recent advancements in LLM-powered code summarization focus on retrieving similar code snippets and leverage LLMs' few-shot learning capability to enhance performance. [154] retrieves similar code examples by measuring token overlap and the cosine distance between embedding vectors of code snippets. In contrast, [51] employs the BM25 algorithm and incorporates repository information, data flow information, and variable information to construct three-shot prompts. SCLA [284] further enhances code semantics in LLM prompts by preprocessing the code sample pool to extract semantic information. By simultaneously leveraging few-shot learning, it achieves state-of-the-art performance based on Gemini-1.5-Pro.

• ***LLM as Repository-Level Code Completer.*** Repository context (e.g., imports, related classes, etc.) plays a crucial role in code completion. Given the strong semantic understanding and generative capabilities of LLMs, how to integrate contextual information into code completion has become a key research focus. RepoFusion [357] appends the surrounding text of the target code to the repository context retrieved based on BM25, encoding and concatenating them as input to the decoder for code generation. This approach enables the model to produce context-aware code completions by leveraging both local and repository-level information. CoCoMIC [118] proposes a more robust retrieval method based on program dependency graphs. Given an incomplete program, it retrieves the most relevant context by analyzing file imports within

the constructed graph. By defining the relevant context as files within a two-hop neighborhood, this approach mitigates the risk of excluding vital dependencies while avoiding the inclusion of irrelevant information. However, some researchers have found that simple retrieval methods fail to improve performance in up to 80% of cases and may even degrade performance due to the inclusion of irrelevant information [413]. As a result, Repoformer introduces a self-supervised learning approach to enable the model to accurately judge whether retrieval can improve its output quality. A new <*eof*> token is introduced to guide the model in determining whether context retrieval is necessary. Based on the output after <*eof*> token, it decides whether to generate the output directly or to perform retrieval first.

3.3 LLM for Data System Optimization

This section presents the application of LLM to optimize the performance of different data systems across three key tasks: *(1) Configuration Tuning:* selecting effective system configurations, such as database knobs and indexes; *(2) Query Optimization:* accelerating input SQL queries through logical rewrites and physical plan selection; *(3) Anomaly Diagnosis:* addressing system anomalies, such as spikes in the usage of specific system resources.

3.3.1 LLM *for Configuration Tuning*

Configuration tuning aims to identify effective configurations, such as database knobs [231], [474] and indexes [485], [487], [486], to optimize the system performance. Traditional tuning approaches, including rule-based methods and learning-based techniques with classical machine learning models, often require extensive explorations without a promising starting point [231]. Furthermore, they might result in sub-optimal configurations, despite using advanced techniques such as transfer learning [463], [402].

A key limitation of these methods is the failure to incorporate extensive domain knowledge (e.g., information from system manuals and public forum discussions) into the tuning process, relying solely on runtime feedback from benchmark evaluations to guide optimization. To address this issue, recent approaches utilize LLM with large-scale domain knowledge to enhance the tuning process via the following methods.

Tuning Task-Aware Prompt Engineering. The first method manually designs prompts with informative details (e.g., system status) to assist LLM in configuration tuning (e.g., database knobs and indexes). Some approaches further enhance this by introducing automatic prompt generation techniques or by formulating it as an optimization problem.

(1) Manually-Crafted Tuning Prompt. Existing methods design prompts that incorporate essential details (e.g., system status) tailored to the characteristics of specific tasks. In particular, the constructed prompts typically consist of the following components.

• ***Configuration Task Instruction.*** To convey the overall tuning objective, existing methods specify task instructions in the prompts using chain-of-thought (CoT) and role-play-based guidance. For instance, LLMBench [243] explicitly defines the goals of three key subtasks in knob tuning: (i) knob pruning to retain the most influential knobs, (ii) model initialization to select promising knobs for warm-starting bayesian optimization, and (iii) knob recommendation to return optimal configurations for specific workloads. Similarly, LATuner [132] instructs LLM to identify critical knobs for warm-starting the tuning process and select promising knobs as training samples for boosting the sampling procedure.

• ***Input Tuning Context.*** To enable LLM to effectively support the tuning process for specific workloads, existing methods enrich the tuning context with detailed information. Specifically, prompts are carefully structured to include: (i) Configuration Specifications: list of tunable knobs (e.g., names and allowable value ranges) and usage descriptions, including fixed-task demonstrations (e.g., LLMBench [243], LATuner [132]); (ii) Environment Information: covering workload and database characteristics (e.g., compressed SQL snippets with join conditions in λ-$Tune$ [156]), as well as hardware settings (e.g., memory size and CPU core count).

• ***Output Tuning Requirement.*** To ensure accurate parsing and interpretation of configurations generated by LLM, output formats are explicitly specified in the prompt. For instance, LLMBench [243] requires that recommended knob values be returned in JSON format, while LATuner [132] enforces constraints such as excluding the use of the "None" value in the configuration output.

(2) Automatic Tuning Prompt Generation. To improve the efficiency of prompt generation for different workloads, existing methods propose the following techniques to automate the process of identifying effective prompts.

• ***Input Specific Prompt Generation.*** To identify the most suitable prompts for varying tasks, existing methods automatically tailor prompt generation based on specific inputs. For example, DB-GPT [491] introduces an automatic prompt generation framework that leverages LLM to produce multiple instruction candidates, selecting the optimal ones using scoring functions associated with the performance improvement. Additionally, DB-GPT [491] and LLMIdxAdvis [473] select demonstration examples in the prompts based on semantic similarity between candidate examples and input queries, as computed by a model-based encoder.

• ***Optimization Problem Formulation.*** To reduce token usage and convey the most relevant context to the LLM, some methods formulate prompt generation as a cost-based optimization problem. For instance, λ-$Tune$ [156] compresses workload representations by modeling the selection of join conditions as an integer linear programming problem, introducing binary decision variables to capture the positional relationships of different columns.

RAG Based Tuning Experience Enrichment. The second method builds an offline knowledge base from diverse external sources and performs online retrieval to provide LLM with context-specific knowledge (e.g., similar historical tuning cases). This approach addresses the limitations of direct prompting, which often yields overly generic responses lacking concrete commands and effective configurations [96].

(1)* LLM *Based Tuning Experience Preparation. Given that existing tuning knowledge is distributed across heterogeneous formats, LLMs are employed to construct a knowledge base by processing and integrating multi-source external experience in an offline manner. For example, GPTuner [223] prompts LLM to extract implicit knowledge, remove noisy content, and summarize relevant information from multiple sources. Additionally, it introduces a prompt ensemble algo-

rithm that generates multiple prompts by varying the demonstration examples, aiming to mitigate hallucination issues.

(2) Semantic Based Tuning Experience Retrieval. To improve the accuracy of relevant experience retrieval, existing methods employ model-based encoders to capture semantic relationships (e.g., documents conveying similar meanings with different expressions). For instance, Andromeda [96] utilizes a Sentence-BERT encoder trained with contrastive learning to generate embeddings, which are then used to perform similarity searches across various sources, including historical queries and troubleshooting manuals.

Training Enhanced Tuning Goal Alignment. The third method introduces additional training to further refine LLMs, improving their alignment with tuning objectives. For example, DB-GPT [491] proposes techniques to facilitate effective fine-tuning, including: (i) heuristic statistical data embedding, (ii) LLM-assisted annotation of high-quality samples, (iii) contrastive learning of supplementary training data generation, and (iv) delta tuning to minimize trainable parameters while maintaining performance. Similarly, E2ETune [177] fine-tunes LLMs (e.g., Mistral-7B) using training data comprising "*(workload)* $\rightarrow$ *(configuration)*" pairs, where diverse workloads are generated via GPT-4 prompting and optimal configurations are identified using the HEBO algorithm [112].

3.3.2 LLM *for Query Optimization*

Query optimization aims to accelerate SQL execution through logical (e.g., query rewriting) and physical (e.g., join order and plan selection) enhancements. Traditional logical optimization relies on predefined rewrite rules or learning-based approaches to determine rule application order, while physical optimization employs heuristic algorithms using statistical data or learning-based techniques leveraging query plan features. However, these approaches often overlook external SQL optimization knowledge, limiting their effectiveness and generalizability across diverse SQL patterns.

To address these limitations, recent studies investigate the use of LLM to directly rewrite input SQL queries or determine optimal rule application sequences for logical optimization. They also explore leveraging LLM to select optimal query execution plans for physical optimization, drawing on the extensive SQL optimization knowledge encoded within the model. These methods can be broadly categorized as follows.

Optimization-Aware Prompt Engineering. The first method directly employs LLMs to perform query optimization using well-structured prompts composed of two key components: (i) manually crafted templates enriched with task-specific details (e.g., explicit task instructions), and (ii) relevant optimization examples automatically selected to more effectively guide the optimization process.

(1) Manually-Crafted Optimization Prompt. Existing methods construct prompts with the following components to facilitate the query optimization task.

• ***Optimization Task Instruction.*** To clarify the optimization objective and guide LLMs to produce specific optimization actions, detailed task instructions are included in the prompts. For logical query optimization, some methods instruct LLMs to directly generate equivalent rewritten queries with improved performance (e.g., DB-GPT [491], GenRewrite [261], and LITHE [363]), while others ask them to determine the optimal sequence of rewrite rule applications for a given query (e.g., LLM-R^2[248] and R-Bot[369]). For physical query optimization, some approaches prompt LLMs to generate complete query plans with specified operators and join orders (e.g., LLM-QO [196]), while others instruct LLMs to generate optimization hints or select the most effective plan from a set of candidates (e.g., LLMOpt [438]).

• ***Input Optimization Context.*** To enable effective query optimization for specific workloads, existing methods augment prompts with additional contextual information to better inform LLMs. This includes: (i) Database Statistics: column selectivity [363], histograms, distinct value counts, and estimated cardinalities [196]; (ii) Rule Specifications: a list of applicable rewrite rules accompanied by usage descriptions (e.g., GenRewrite [261] presents natural language hints as the rules) and illustrative examples [248].

• ***Output Optimization Requirement.*** To ensure that the optimizations produced by LLMs are valid and easily processed for downstream use, some methods explicitly define output formatting requirements within the prompts. For example, LLM-R^2 enforces that selected rewrite rules be returned in the format "rules selected: [rule names]" [248], while LLM-QO specifies that the generated query plan should follow the "join operator(table1, table2)" format [196].

(2) In-Context Learning with Optimization Example. Rather than relying on fixed examples to illustrate how LLM should perform optimization, some methods automatically retrieve examples that are semantically similar to the input query to provide more effective guidance. For instance, LLM-R^2 [248] introduces a contrastive representation model to encode query plans based on features such as operators, cardinalities, and costs, and retrieves a set of high-quality demonstrations, i.e., successfully optimized rewritten queries.

RAG Based Optimization Experience Enrichment. The second method adopts the retrieval-augmented generation (RAG) paradigm to equip LLM with relevant contextual information for targeted optimization of specific queries. It constructs and retrieves optimization knowledge from multiple sources that are semantically related to the input query.

(1)* LLM *Based Optimization Experience Preparation. To consolidate optimization experience from multiple sources, existing methods introduce an offline preparation pipeline that leverages LLM to process and integrate data into a unified format. For example, R-Bot [369] employs LLM to generate rewrite rule specifications by (i) summarizing rule code within a hierarchical structure and (ii) extracting information from structured documentation blocks. It further uses LLM to standardize the resulting specifications, explicitly outlining application conditions and detailed rewrite transformations.

(2) Hybrid Optimization Experience Retrieval. To more accurately identify relevant optimization experiences, both structural and semantic characteristics of the input queries are considered during similarity search. For instance, R-Bot [369] introduces a hybrid retrieval approach that computes similarity using concatenated embeddings capturing structural features (e.g., rewrite rule explanations) and semantic representations (e.g., query template structures). Based on the retrieved experience, R-Bot employs a step-by-step LLM-driven rewrite process, further enhanced through a self-reflection mechanism to improve rewrite quality.

Training Enhanced Optimization Improvement. The third method either uses LLM outputs to train smaller models

or fine-tunes LLMs on task-specific data to support various query optimization tasks (e.g., query plan generation). For instance, LLMSteer [53] uses LLM-generated embeddings to train a classifier for selecting optimal hints of the input SQL. LLM-QO [196] fine-tunes LLMs to generate execution plans directly through a two-stage pipeline: (i) Query Instruction Tuning (QIT) for producing valid plans; (ii) Query Direct Preference Optimization (QDPO) for distinguishing high-quality plans. The fine-tuning data is structured as "*(query, task instruction, auxiliary information such as schema and statistics, demonstration)*" paired with the corresponding efficient execution plan. LLMOpt [438] fine-tunes two models: (i) LLMOpt(G), which generates candidate hints, and (ii) LLMOpt(S), which selects the optimal hint as a list-wise cost model. The fine-tuning data is structured as "*(query, statistics such as histograms)* → *(optimal hint)*" for LLMOpt(G) and "*(query, statistics such as histograms, candidate hints)* → *(index of optimal hint)*" for LLMOpt(S).

3.3.3 LLM *for Anomaly Diagnosis*

Anomaly diagnosis focuses on analyzing root causes and identifying recovery solutions for anomalies (e.g., spikes in system resource usage) during the system runtime, such as databases. Traditional rule-based methods often fail to accurately identify root causes across diverse scenarios, while classical machine learning models (e.g., random forests) cannot generate comprehensive reports with detailed recovery solutions.

Recent studies demonstrate that LLMs, with their advanced textual understanding and reasoning capabilities, can effectively pinpoint root causes and generate detailed diagnosis reports with recovery solutions in various formats. These LLM-based approaches can be categorized as follows.

Manually Crafted Prompts for Anomaly Diagnosis. The first method emulates the reasoning process of a human DBA, which involves referencing essential statistical information and conducting an in-depth analysis during diagnosis. The information is incorporated into well-structured prompts to enhance diagnosis accuracy. For example, DBG-PT [155] utilizes LLM to detect query execution slowdowns caused by changes in query plans, using prompts that include: (i) a summary of plan differences, (ii) a request for feasible configuration recommendations, and (iii) a specification of the reasoning process with output formatted in JSON format.

RAG Based Diagnosis Experience Enrichment. The second method adopts retrieval-augmented generation (RAG) paradigm to provide LLM with relevant diagnosis knowledge, leveraging two key components: a knowledge base and a retriever. For instance, D-Bot [490], [489] enhances database anomaly diagnosis by preparing a corpus of documents and tools considering the hierarchical document structure, then using a fine-tuned Sentence-BERT encoder to retrieve relevant materials and guide LLM via prompts enriched with the retrieved content. ByteHTAP [425] supports LLM-based diagnosis of query performance regressions in HTAP systems by first constructing a knowledge base of historical queries and their associated performance explanations. It then employs an enhanced tree-CNN classifier to encode and retrieve relevant plan pairs. The retrieved information is incorporated into prompts that include: (i) background information (e.g., key differences among HTAP system engines), (ii) a task description (e.g., retrieved diagnosis knowledge with explicit input-output specifications), and (iii) additional user-provided context (e.g., recent index changes).

Multi-Agent Mechanism for Collaborative Diagnosis. The third method adopts an agent-based diagnosis framework, where specialized agents with distinct responsibilities collaborate to improve diagnosis accuracy and efficiency. For example, D-Bot [490], [489] orchestrates multiple domain-specific LLM agents, each aligned with a cluster of pre-processed diagnosis knowledge, to support precise anomaly diagnosis in databases. These agents, coordinated by a chief agent, conduct multi-step root cause analysis via a tree-search algorithm. Similarly, Panda [359] emulates experienced database engineers by leveraging LLM agents across five functional components: (i) question verification to eliminate irrelevant queries, (ii) grounding to provide necessary input query context, (iii) verification to ensure diagnosis accuracy and source attribution, (iv) feedback integration to incorporate user input, and (v) affordance assessment to estimate the performance impact of generated solutions.

Localized LLM Enhancement via Specialized Fine-Tuning. The last method employs specialized fine-tuning strategies for localized LLMs of modest scale (e.g., 6B-14B), leveraging distilled knowledge to approximate the outputs of larger models while achieving comparable performance. For instance, D-Bot [490] applies multi-task fine-tuning to improve the diagnosis capabilities of localized LLMs. Specifically, three models (i.e., Llama2-13B, CodeLlama-13B, and Baichuan2-13B) are fine-tuned to replicate the diagnosis results generated by the GPT-4-powered D-Bot. The fine-tuning dataset consists of samples covering D-Bot diagnosis workflows across five sub-tasks (e.g., tool invocation), along with associated prompts and historical dialogue messages.

Practices of LLMs for Data Management

Alibaba Cloud [5] has integrated Text-to-SQL features into its BI platform, facilitating NL queries over structured datasets. Amazon Nova [3] employs automated document processing to extract structured information from diverse unstructured sources. In terms of data systems, PawSQL [41], an advanced query optimization platform, offers both SQL rewriting and index recommendation capabilities, adopted by over 10,000 professionals. Database diagnosis also thrives on a robust ecosystem. For instance, DB-Doctor [35], compatible with mainstream databases, delivers kernel-level performance diagnostics for comprehensive system analysis and optimization.

4 Challenges and Future Directions

4.1 Data Management for LLM

4.1.1 Task-Specific Data Selection for Efficient Pretraining

In LLM pre-training, vast amounts of general data are typically used, but much of this data may not be relevant to the target task. The inclusion of irrelevant data not only increases training time but also impedes the model's adaptability to specific tasks. For instance, when training a model for the medical domain, unrelated data sources such as news

articles and social media posts may hinder the learning of domain-specific knowledge. Consequently, the challenge lies in automatically selecting task-relevant data while discarding irrelevant information during pretraining. Currently, most approaches rely on hand-crafted filtering rules or fixed labeled datasets for data selection, lacking dynamic strategies that adapt to the model's evolving task-specific needs. Exploring methods to automatically select relevant data and discard irrelevant data during pre-training represents a promising avenue for improving task adaptability and training efficiency.

4.1.2 Optimizing Data Processing Pipelines

Currently, the construction of data processing pipelines for LLMs relies heavily on experience and experimentation. For instance, in building the FineWeb dataset, decisions such as whether to use the WET or WARC format for text extraction from CommonCrawl, or whether to apply a global MinHash approach for deduplication or perform it separately for each snapshot, are made only after training models and benchmarking their performance. However, this experimental methodology is resource-intensive. In the case of FineWeb, over 70 models with 1 billion parameters were trained, consuming a total of 80,000 H100 GPU hours. To improve the efficiency of these pipelines, future research should focus on developing data-driven methods that can predict optimal preprocessing configurations. in advance, reducing the reliance on costly trial-and-error approaches. This would not only minimize computational costs but also accelerate the development of high-quality datasets for LLMs.

4.1.3 LLM Knowledge Update and Version Control

In fast-evolving domains (e.g., healthcare, finance, law), knowledge is constantly updated. To ensure the reliability of LLMs, the data used for training and fine-tuning must be up-to-date. Delays in incorporating the latest knowledge can result in outdated or harmful outputs, particularly in fields like medicine where guidelines frequently change. While there have been various approaches to data synthesis and augmentation, little attention has been given to efficiently managing rapid knowledge updates or resolving contradictions when new information conflicts with older data. Existing systems often rely on static datasets, which are problematic in dynamic sectors. Although platforms like ChatGPT and Deepseek allow LLMs to search the web, this approach may not always guarantee accuracy or relevance, leading to suboptimal results. A more effective solution would involve a platform that facilitates the creation, sharing, and version control of datasets with real-time knowledge updates. By leveraging community-driven contributions, this platform could enable users to synthesize and share datasets using customizable methods, such as LLM-generated prompts from documents or websites, offering continuous, high-quality updates and improving the overall accuracy and reliability of LLMs.

4.1.4 Comprehensive Dataset Evaluation

The performance enhancement of models is closely tied to the use of 'high-quality' datasets. However, determining what constitutes a high-quality dataset remains a challenge. Typically, the quality of a dataset can only be inferred after training and evaluating a model, which makes the process indirect and resource-intensive. When a dataset's quality is subpar, it can lead to significant computational overhead and inefficiencies. While existing research [393] has proposed a model-agnostic method for evaluating datasets across three aspects: reliability, difficulty, and validity. These dimensions alone do not fully capture a dataset's quality. The current framework falls short of providing a comprehensive evaluation that aligns with the model's capabilities and performance improvements. Therefore, a promising direction for future research is the development of a robust dataset evaluation system that does not rely on model training. This system should provide consistent quality scores that directly correlate with model performance enhancements, enabling more efficient dataset selection and use without the need for exhaustive training cycles.

4.1.5 Hybrid RAG Indexing and Retrieval

Currently, there lacks a single database that integrates full-text, vector, knowledge graph, and structured search interfaces into a cohesive indexing and retrieval engine for Retrieval-Augmented Generation (RAG) training. While systems like Elasticsearch [36] excel in full-text and vector search, and LightRAG [164] has introduced advanced vector and graph processing, these solutions remain siloed. They lack a unified platform designed specifically for hybrid RAG, where multiple indexing and search mechanisms coexist to support efficient downstream applications. Although emerging platforms like AutoRAG [209] provide frameworks for constructing RAG pipelines, they focus on workflow management, model integration, and automation rather than offering a fully integrated database with indexing and retrieval engines. A promising direction for future RAG data serving is the development of an integrated platform that provides seamless indexing and retrieval for diverse data types, while also integrating data serving features such as knowledge filtering and re-ranking [47], thereby improving the efficiency and flexibility of RAG applications.

4.2 LLM for Data Management

4.2.1 Unified Data Analysis System

One of the major challenges in LLM for Data Analysis is the absence of a unified system capable of handling diverse data types. Currently, analyzing different data formats often requires designing task-specific models separately. The most straightforward approach to enabling a system to process all types of data is to integrate these models into a single framework. However, this leads to prohibitively high deployment and maintenance costs due to the need to manage multiple models simultaneously. A more promising direction is to develop a model that can flexibly accommodate various data inputs and user requirements while supporting the analysis of structured, semi-structured, and unstructured data. Such a system would establish a paradigm for LLM for Data Analysis at the system level and offer a generalized capability for analyzing data across different structural types, thereby facilitating data automation.

4.2.2 Data Analysis with Private Domain Knowledge

Another challenge in leveraging LLMs for data analysis is the effective utilization of private domain knowledge. Current

approaches primarily rely on RAG to retrieve relevant knowledge or fine-tune models on domain-specific datasets. However, these methods struggle when dealing with novel or highly complex domain knowledge. For example, in Text-to-SQL tasks involving large-scale databases with 10,000 columns and 1,000,000 rows, where each column is associated with specific domain knowledge, existing techniques often fail to generalize effectively. The lack of datasets that explicitly incorporate domain knowledge further exacerbates this issue, making it difficult to meet the demands of real-world industrial applications. Consequently, developing more advanced mechanisms for integrating domain knowledge into LLMs remains a critical open research problem.

4.2.3 Representing Non-Sequential and Non-Textual Data

Current LLM-based approaches typically transform non-sequential and non-textual data into serialized textual formats to align with the input requirements of LLMs [129], [196], [438]. While this enables basic compatibility, it overlooks the original structural semantics of the data and can lead to significant information loss in downstream tasks. For instance, in data manipulation and analysis, relational tables (originally structured as two-dimensional matrices) are typically flattened into multiple serialized sequences, obscuring inherent row-column relationships [78], [74], [319]. Similarly, in system optimization tasks, crucial statistical signals such as column selectivities and histograms are either omitted or naively encoded as plain texts, limiting their utility in guiding optimization decisions [156], [132]. Consequently, a promising future direction is to develop more expressive and task-aware representations that preserve the structural and statistical integrity of such data. This includes leveraging multi-modal LLMs or designing tailored encoding strategies that maintain the uniqueness of these data types, thereby enabling more effective and semantically informed LLM applications.

4.2.4 Efficient LLM Utilization Under Budget Constraints

While LLMs have shown strong potential across data manipulation, analysis, and system optimization tasks, their high computational cost and latency pose challenges for real-time or large-scale applications [196], [53]. For example, relying solely on LLMs is impractical for processing tens of millions of rows in relational table analysis due to prohibitive resource demands [432], [304]. Similarly, current LLM-based query optimizers often require minutes per query, far exceeding the millisecond-level efficiency of traditional statistical methods [369], [248]. Therefore, a promising direction is to develop hybrid strategies that integrate LLMs with traditional techniques or to devise scheduling mechanisms that allocate tasks across multiple LLMs based on cost-performance trade-offs. Such approaches can enhance the practicality and scalability of LLM-based systems under real-world budget constraints.

5 Conclusion

In this paper, we summarize the recent techniques on DATA4LLM and LLM4DATA. The former focuses on utilizing data processing, storage, serving techniques to address the data problems in different LLM stages. The latter focuses on using LLM capabilities to reduce the complexity of conducting data management, e.g., data manipulation, data analysis, and data system optimization. We also provide some research challenges and open problems in DATA4LLM, LLM4DATA, and hybrid data and LLM optimization.

References

[1] https://arangodb.com/.
[2] https://arxiv.org/.
[3] https://aws.amazon.com/cn/ai/generative-ai/nova/understanding/.
[4] https://aws.amazon.com/s3.
[5] https://bailian.console.aliyun.com/xiyan.
[6] https://beautiful-soup-4.readthedocs.io/en/latest/.
[7] https://bitbucket.org/product/.
[8] https://blazegraph.com/.
[9] https://cachelib.org/.
[10] https://cocodataset.org/.
[11] https://commoncrawl.org/.
[12] https://docs.cohere.com.
[13] https://docs.python.org/3/library/pickle.html.
[14] https://github.com/.
[15] https://github.com/deepseek-ai/3fs.
[16] https://github.com/juicedata/juicefs.
[17] https://github.com/neo4j/neo4j.
[18] https://github.com/paddlepaddle/paddleocr.
[19] https://github.com/seleniumhq/selenium.
[20] https://gitlab.com/.
[21] https://graphdb.ontotext.com/.
[22] https://huggingface.co/.
[23] https://huggingface.co/ckiplab/bert-tiny-chinese.
[24] https://huggingface.co/infgrad/stella-large-zh-v2.
[25] https://lancedb.com.
[26] https://milvus.io.
[27] https://onnx.ai.
[28] https://openlibrary.org/.
[29] https://paddlenlp.readthedocs.io.
[30] https://playwright.dev/.
[31] https://pptr.dev/.
[32] https://pytorch.org/.
[33] https://spacy.io/.
[34] https://weaviate.io.
[35] https://www.dbdoctor.cn/.
[36] https://www.elastic.co/elasticsearch.
[37] https://www.eyelevel.ai/post/do-vector-databases-lose-accuracy-at-scale.
[38] https://www.gutenberg.org/.
[39] https://www.llamaindex.ai/.
[40] https://www.mindspore.cn/.
[41] https://www.pawsql.com/.
[42] https://www.tensorflow.org.
[43] https://www.tensorflow.org/guide/data.
[44] https://www.tensorflow.org/tutorials/load_data/tfrecord.
[45] A. Abbas, E. Rusak, K. Tirumala, W. Brendel, K. Chaudhuri, and A. S. Morcos. Effective pruning of web-scale datasets based on complexity of concept clusters. *arXiv preprint arXiv:2401.04578*, 2024.
[46] A. Abbas, K. Tirumala, D. Simig, S. Ganguli, and A. S. Morcos. Semdedup: Data-efficient learning at web-scale through semantic deduplication. *arXiv preprint arXiv:2303.09540*, 2023.
[47] A. Abdallah, J. Mozafari, B. Piryani, and A. Jatowt. Asrank: Zero-shot re-ranking with answer scent for document retrieval. *arXiv preprint arXiv:2501.15245*, 2025.
[48] S. Abiteboul. Querying semi-structured data. In *Database Theory—ICDT'97: 6th International Conference Delphi, Greece, January 8–10, 1997 Proceedings 6*, pages 1–18. Springer, 1997.
[49] K. Aggarwal, A. Khandelwal, K. Tanmay, O. M. Khan, Q. Liu, M. Choudhury, H. H. Chauhan, S. Som, V. Chaudhary, and S. Tiwary. Dublin: Visual document understanding by language-image network, 2023.
[50] C. Aguerrebere, I. Bhati, M. Hildebrand, M. Tepper, and T. Willke. Similarity search in the blink of an eye with compressed indices, 2023.

[51] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr. Automatic semantic augmentation of language model prompts (for code summarization). In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[52] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics (demonstrations)*, pages 54–59, 2019.

[53] P. Akioyamen, Z. Yi, and R. Marcus. The unreasonable effectiveness of llms for query optimization. *CoRR*, abs/2411.02862, 2024.

[54] M. M. Alam and W. Wang. A comprehensive survey on data provenance: State-of-the-art approaches and their deployments for iot security enforcement. *J. Comput. Secur.*, 29(4):423–446, 2021.

[55] A. Albalak, Y. Elazar, S. M. Xie, S. Longpre, N. Lambert, X. Wang, N. Muennighoff, B. Hou, L. Pan, H. Jeong, et al. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827*, 2024.

[56] A. Albalak, L. Pan, C. Raffel, and W. Y. Wang. Efficient online data mixing for language model pre-training. In *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*, 2023.

[57] K. An, F. Yang, L. Li, J. Lu, S. Cheng, S. Si, L. Wang, P. Zhao, L. Cao, Q. Lin, et al. Thread: A logic-based data organization paradigm for how-to question answering with retrieval augmented generation. *arXiv preprint arXiv:2406.13372*, 2024.

[58] Q. An, C. Ying, Y. Zhu, Y. Xu, M. Zhang, and J. Wang. LEDD: large language model-empowered data discovery in data lakes. *CoRR*, abs/2502.15182, 2025.

[59] R. Angles. A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177, 2012.

[60] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), Feb. 2008.

[61] Z. Ankner, C. Blakeney, K. Sreenivasan, M. Marion, M. L. Leavitt, and M. Paul. Perplexed by perplexity: Perplexity-based data pruning with small reference models. *arXiv preprint arXiv:2405.20541*, 2024.

[62] S. Appalaraju, P. Tang, Q. Dong, N. Sankaran, Y. Zhou, and R. Manmatha. Docformerv2: Local features for document understanding, 2023.

[63] S. Arora, B. Yang, S. Eyuboglu, A. Narayan, A. Hojel, I. Trummer, and C. Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proc. VLDB Endow.*, 17(2):92–105, 2023.

[64] M. Artetxe, S. Bhosale, N. Goyal, T. Mihaylov, M. Ott, S. Shleifer, X. V. Lin, J. Du, S. Iyer, R. Pasunuru, et al. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*, 2021.

[65] G. A. Atemezing. Empirical evaluation of a cloud-based graph database: the case of neptune. In *Knowledge Graphs and Semantic Web: Third Iberoamerican Conference and Second Indo-American Conference, KGSWC 2021, Kingsville, Texas, USA, November 22–24, 2021, Proceedings 3*, pages 31–46. Springer, 2021.

[66] J.-M. Attendu and J.-P. Corbeil. Nlu on data diets: Dynamic data subset selection for nlp classification tasks. *arXiv preprint arXiv:2306.03208*, 2023.

[67] A. Audibert, Y. Chen, D. Graur, A. Klimovic, J. Šimša, and C. A. Thekkath. tf. data service: A case for disaggregating ml input data processing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, pages 358–375, 2023.

[68] T. Ayoola, S. Tyagi, J. Fisher, C. Christodoulopoulos, and A. Pierleoni. Refined: An efficient zero-shot-capable approach to end-to-end entity linking, 2022.

[69] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[70] S. Bai, K. Chen, X. Liu, et al. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923, 2025.

[71] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[72] M. I. L. Balaka, D. Alexander, Q. Wang, Y. Gong, A. Krisnadhi, and R. C. Fernandez. Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. *arXiv preprint arXiv:2504.09207*, 2025.

[73] A. Barbaresi. Trafilatura: A web scraping library and command-line tool for text discovery and extraction. In H. Ji, J. C. Park, and R. Xia, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 122–131, Online, Aug. 2021. Association for Computational Linguistics.

[74] T. Bendinelli, A. Dox, and C. Holz. Exploring llm agents for cleaning tabular machine learning datasets. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025. arXiv:2503.06664.

[75] M. Berglund and B. van der Merwe. Formalizing bpe tokenization. *arXiv preprint arXiv:2309.08715*, 2023.

[76] J. Bevendorff, S. Gupta, J. Kiesel, and B. Stein. An empirical comparison of web content extraction algorithms. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2594–2603, New York, NY, USA, 2023. Association for Computing Machinery.

[77] S. Biderman, U. Prashanth, L. Sutawika, H. Schoelkopf, Q. Anthony, S. Purohit, and E. Raff. Emergent and predictable memorization in large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[78] F. Biester, M. Abdelaal, and D. D. Gaudio. Llmclean: Context-aware tabular data cleaning via llm-generated ofds. In *ADBIS (Short Papers)*, volume 2186 of *Communications in Computer and Information Science*, pages 68–78. Springer, 2024.

[79] D. Borthakur et al. Hdfs architecture guide. *Hadoop apache project*, 53(1-13):2, 2008.

[80] D. Brandfonbrener, H. Zhang, A. Kirsch, J. R. Schwarz, and S. Kakade. Color-filter: Conditional loss reduction filtering for targeted language model pre-training. *arXiv preprint arXiv:2406.10670*, 2024.

[81] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[82] L. Cao. Tablemaster: A recipe to advance table understanding with language models, 2025.

[83] Q. Cao, M. Najibi, and S. Mehta. Ctrlsynth: Controllable image text synthesis for data-efficient multimodal learning, 2024.

[84] Y. Cao, Y. Kang, C. Wang, and L. Sun. Instruction mining: Instruction data selection for tuning large language models. *arXiv preprint arXiv:2307.06290*, 2023.

[85] B. Casey, K. Damian, A. Cotaj, and J. C. S. Santos. An empirical study of safetensors' usage trends and developers' perceptions, 2025.

[86] C. Chai, J. Wang, Y. Luo, Z. Niu, and G. Li. Data management for machine learning: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(5):4646–4667, 2023.

[87] C.-Y. Chang, Z. Jiang, V. Rakesh, M. Pan, et al. Main-rag: Multi-agent filtering retrieval-augmented generation, 2024.

[88] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[89] T.-Y. Che, X.-L. Mao, T. Lan, and H. Huang. A hierarchical context augmentation method to improve retrieval-augmented llms on scientific papers. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 243–254, 2024.

[90] D. Chen, Y. Huang, Z. Ma, H. Chen, X. Pan, C. Ge, D. Gao, Y. Xie, Z. Liu, J. Gao, et al. Data-juicer: A one-stop data processing system for large language models. In *Companion of the 2024 International Conference on Management of Data*, pages 120–134, 2024.

[91] D. Chen, H. Wang, Y. Huang, C. Ge, Y. Li, B. Ding, and J. Zhou. Data-juicer sandbox: A feedback-driven suite for multimodal data-model co-development. *arXiv preprint arXiv:2407.11784*, 2024.

[92] H. Chen, A. Waheed, X. Li, Y. Wang, J. Wang, B. Raj, and M. I. Abdin. On the diversity of synthetic data and its impact on training large language models. *arXiv preprint arXiv:2410.15226*, 2024.

[93] J. Chen, Z. Chen, J. Wang, K. Zhou, Y. Zhu, J. Jiang, Y. Min, W. X. Zhao, Z. Dou, J. Mao, et al. Towards effective and efficient continual pre-training of large language models. *arXiv preprint arXiv:2407.18743*, 2024.

[94] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu. M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *ACL*, 2024.

[95] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[96] S. Chen, J. Fan, B. Wu, N. Tang, C. Deng, P. Wang, Y. Li, J. Tan, F. Li, J. Zhou, and X. Du. Automatic database configuration debugging using retrieval-augmented language models. *CoRR*, abs/2412.07548, 2024.

[97] T. Chen, H. Wang, S. Chen, W. Yu, K. Ma, X. Zhao, H. Zhang, and D. Yu. Dense x retrieval: What retrieval granularity should we use? *arXiv preprint arXiv:2312.06648*, 2023.

[98] Z. Chen, T. Liu, M. Tian, W. Luo, Z. Liu, et al. Advancing mathematical reasoning in language models: The impact of problem-solving data, data synthesis methods, and training stages. In *The Thirteenth International Conference on Learning Representations*, 2025.

[99] D. Cheng, Y. Gu, S. Huang, J. Bi, M. Huang, and F. Wei. Instruction pre-training: Language models are supervised multitask learners. *arXiv preprint arXiv:2406.14491*, 2024.

[100] S. Cheng, Z. Zhuang, Y. Xu, F. Yang, C. Zhang, X. Qin, X. Huang, L. Chen, Q. Lin, D. Zhang, S. Rajmohan, and Q. Zhang. Call me when necessary: Llms can efficiently and faithfully reason over structured environments, 2024.

[101] X. Cheng, X. Wang, X. Zhang, T. Ge, S.-Q. Chen, F. Wei, H. Zhang, and D. Zhao. xrag: Extreme context compression for retrieval-augmented generation with one token. *arXiv preprint arXiv:2405.13792*, 2024.

[102] A. Chevalier, A. Wettig, A. Ajith, and D. Chen. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*, 2023.

[103] J. Choi, J. Yun, K. Jin, and Y. Kim. Multi-news+: Cost-efficient dataset cleansing via llm-based data annotation. In *EMNLP*, pages 15–29. Association for Computational Linguistics, 2024.

[104] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022.

[105] M. Christ, S. Gunn, and O. Zamir. Undetectable watermarks for language models. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1125–1139. PMLR, 2024.

[106] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[107] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[108] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153:235–256, 2007.

[109] T. Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023.

[110] M. Conover, M. Hayes, A. Mathur, J. Xie, J. Wan, S. Shah, A. Ghodsi, P. Wendell, M. Zaharia, and R. Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023.

[111] A. Cossu, A. Carta, L. Passaro, V. Lomonaco, T. Tuytelaars, and D. Bacciu. Continual pre-training mitigates forgetting in language and vision. *Neural Networks*, 179:106492, 2024.

[112] A. I. Cowen-Rivers, W. Lyu, Z. Wang, R. Tutunov, J. Hao, J. Wang, and H. Bou-Ammar. HEBO: heteroscedastic evolutionary bayesian optimisation. *CoRR*, abs/2012.03826, 2020.

[113] G. Cui, L. Yuan, N. Ding, G. Yao, B. He, W. Zhu, Y. Ni, G. Xie, R. Xie, Y. Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. In *Forty-first International Conference on Machine Learning*, 2024.

[114] J. Cui, Z. Li, Y. Yan, B. Chen, and L. Yuan. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *CoRR*, abs/2306.16092, 2023.

[115] Y. Dai, D. Feng, J. Huang, H. Jia, Q. Xie, Y. Zhang, W. Han, W. Tian, and H. Wang. Laiw: A chinese legal large language models benchmark, 2024.

[116] H. Ding, Z. Wang, G. Paolini, V. Kumar, A. Deoras, D. Roth, and S. Soatto. Fewer truncations improve language modeling. *arXiv preprint arXiv:2404.10830*, 2024.

[117] N. Ding, Y. Chen, B. Xu, Y. Qin, Z. Zheng, S. Hu, Z. Liu, M. Sun, and B. Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*, 2023.

[118] Y. Ding, Z. Wang, W. Ahmad, M. K. Ramanathan, R. Nallapati, P. Bhatia, D. Roth, and B. Xiang. CoCoMIC: Code Completion by Jointly Modeling In-file and Cross-file Context. In N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 3433–3445, Torino, Italia, May 2024. ELRA and ICCL.

[119] Y. Ding, Q. Zeng, and T. Weninger. Chatel: Entity linking with chatbots, 2024.

[120] J. Dodge, M. Sap, A. Marasović, W. Agnew, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[121] F. Dong, Y. He, Y. Liang, Z. Liu, Y. Wu, P. Chen, and T. Yang. Simisketch: Efficiently estimating similarity of streaming multisets. *arXiv preprint arXiv:2405.19711*, 2024.

[122] G. Dong, D. Pan, Y. Sun, S. Zhang, Z. Liang, X. Wu, Y. Shen, F. Yang, H. Sun, T. Li, et al. Baichuanseed: Sharing the potential of extensive data collection and deduplication by introducing a competitive large language model baseline. *arXiv preprint arXiv:2408.15079*, 2024.

[123] G. Dong, H. Yuan, K. Lu, C. Li, M. Xue, D. Liu, W. Wang, Z. Yuan, C. Zhou, and J. Zhou. How abilities in large language models are affected by supervised fine-tuning data composition. *arXiv preprint arXiv:2310.05492*, 2023.

[124] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[125] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[126] Q. Du, C. Zong, and J. Zhang. Mods: Model-oriented data selection for instruction tuning. *arXiv preprint arXiv:2311.15653*, 2023.

[127] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[128] M. Eibich, S. Nagpal, and A. Fred-Ojala. Aragog: Advanced rag output grading. *arXiv preprint arXiv:2404.01037*, 2024.

[129] M. Y. Eltabakh, Z. A. Naeem, M. S. Ahmad, M. Ouzzani, and N. Tang. Retclean: Retrieval-based tabular data cleaning using llms and data lakes. *Proc. VLDB Endow.*, 17(12):4421–4424, 2024.

[130] L. Engstrom, A. Feldmann, and A. Madry. Dsdm: Model-aware dataset selection with datamodels. *arXiv preprint arXiv:2401.12926*, 2024.

[131] J. Fairoze, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, and M. Wang. Publicly detectable watermarking for language models. *arXiv preprint arXiv:2310.18491*, 2023.

[132] C. Fan, Z. Pan, W. Sun, C. Yang, and W. Chen. Latuner: An llm-enhanced database tuning system based on adaptive surrogate model. In *ECML/PKDD (5)*, volume 14945 of *Lecture Notes in Computer Science*, pages 372–388. Springer, 2024.

[133] L. Fan, D. Krishnan, P. Isola, D. Katabi, and Y. Tian. Improving clip training with language rewrites, 2023.

[134] M. Fan, X. Han, J. Fan, C. Chai, N. Tang, G. Li, and X. Du. Cost-effective in-context learning for entity resolution: A design space exploration. In *ICDE*, pages 3696–3709. IEEE, 2024.

[135] S. Fan, M. Pagliardini, and M. Jaggi. Doge: Domain reweighting with generalization estimation. *arXiv preprint arXiv:2310.15393*, 2023.

[136] T. Fan, J. Wang, X. Ren, and C. Huang. Minirag: Towards extremely simple retrieval-augmented generation. *arXiv preprint arXiv:2501.06713*, 2025.

[137] M. Fayyaz, E. Aghazadeh, A. Modarressi, M. T. Pilehvar, Y. Yaghoobzadeh, and S. E. Kahou. Bert on a data diet: Finding important examples by gradient-based pruning. *arXiv preprint arXiv:2211.05610*, 2022.

[138] H. Feng, Q. Liu, H. Liu, J. Tang, W. Zhou, H. Li, and C. Huang. Docpedia: Unleashing the power of large multimodal model in the frequency domain for versatile document understanding, 2024.

[139] S. Feng, S. Prabhumoye, K. Kong, D. Su, M. Patwary, M. Shoeybi, and B. Catanzaro. Maximize your data's potential: Enhancing llm accuracy with two-phase pretraining. *arXiv preprint arXiv:2412.15285*, 2024.

[140] W. Feng, C. Hao, Y. Zhang, Y. Han, and H. Wang. Mixture-of-loras: An efficient multitask tuning for large language models. *arXiv preprint arXiv:2403.03432*, 2024.

[141] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou. Codebert: A pre-trained model for programming and natural languages, 2020.

[142] S. Ferré. First steps of an approach to the arc challenge based on descriptive grid models and the minimum description length principle. *arXiv preprint arXiv:2112.00848*, 2021.

[143] B. Feuer, Y. Liu, C. Hegde, and J. Freire. Archetype: A novel framework for open-source column type annotation using large language models. *Proc. VLDB Endow.*, 17(9):2279–2292, 2024.

[144] A. Finn, N. Kushmerick, and B. Smyth. Fact or fiction: Content classification for digital libraries. In *DELOS Workshops / Conferences*, 2001.

[145] M. Fuest, P. Ma, M. Gui, J. Schusterbauer, V. T. Hu, and B. Ommer. Diffusion models and representation learning: A survey, 2024.

[146] S. Y. Gadre, G. Ilharco, A. Fang, J. Hayase, G. Smyrnis, T. Nguyen, R. Marten, M. Wortsman, D. Ghosh, J. Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *Advances in Neural Information Processing Systems*, 36:27092–27112, 2023.

[147] S. Gandhi, M. Zhao, A. Skiadopoulos, and C. Kozyrakis. Recycle: Resilient training of large dnns using pipeline adaptation. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pages 211–228, 2024.

[148] B. Gao, Z. He, P. Sharma, Q. Kang, D. Jevdjic, J. Deng, X. Yang, Z. Yu, and P. Zuo. {Cost-Efficient} large language model serving for multi-turn conversations with {CachedAttention}. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 111–126, 2024.

[149] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[150] S. Gao, Y. Chen, and J. Shu. Fast state restoration in llm serving with hcache, 2024.

[151] S. Gao, C. Gao, Y. He, J. Zeng, L. Nie, X. Xia, and M. Lyu. Code structure–guided transformer for source code summarization. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–32, Jan. 2023.

[152] C. Ge, Z. Ma, D. Chen, Y. Li, and B. Ding. Bimix: A bivariate data mixing law for language model pretraining, 2025.

[153] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021.

[154] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, and X. Liao. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning, 2023.

[155] V. Giannakouris and I. Trummer. DBG-PT: A large language model assisted query performance regression debugger. *Proc. VLDB Endow.*, 17(12):4337–4340, 2024.

[156] V. Giannankouris and I. Trummer. λ-tune: Harnessing large language models for automated database system tuning. *CoRR*, abs/2411.03500, 2024.

[157] S. Goyal, P. Maini, Z. C. Lipton, A. Raghunathan, and J. Z. Kolter. Scaling laws for data filtering–data curation cannot be compute agnostic. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22702–22711, 2024.

[158] D. Graur, D. Aymon, D. Kluser, T. Albrici, C. A. Thekkath, and A. Klimovic. Cachew: Machine learning input data processing as a service. In *2022 usenix annual technical conference (usenix atc 22)*, pages 689–706, 2022.

[159] D. Graur, O. Mraz, M. Li, S. Pourghannad, C. A. Thekkath, and A. Klimovic. Pecan:{Cost-Efficient}{ML} data preprocessing with automatic transformation ordering and hybrid placement. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 649–665, 2024.

[160] J. Gu, Z. Yang, C. Ding, R. Zhao, and F. Tan. Cmr scaling law: Predicting critical mixture ratios for continual pre-training of language models. *arXiv preprint arXiv:2407.17467*, 2024.

[161] R. Gu, K. Zhang, Z. Xu, Y. Che, B. Fan, H. Hou, H. Dai, L. Yi, Y. Ding, G. Chen, et al. Fluid: Dataset abstraction and elastic acceleration for cloud-native deep learning training jobs. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2182–2195. IEEE, 2022.

[162] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[163] Y. Guo, Z. Hu, Y. Mao, B. Zheng, Y. Gao, and M. Zhou. Birdie: Natural language-driven table discovery using differentiable search index, 2025.

[164] Z. GUO, L. Xia, Y. Yu, T. Ao, and C. Huang. LightRAG: Simple and fast retrieval-augmented generation, 2024.

[165] V. Gupta, P. Kandoi, M. B. Vora, S. Zhang, Y. He, R. Reinanda, and V. Srikumar. Temptabqa: Temporal question answering for semi-structured tables, 2023.

[166] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[167] N. He, W. Xiong, H. Liu, Y. Liao, L. Ding, K. Zhang, G. Tang, X. Han, and W. Yang. Softdedup: an efficient data reweighting method for speeding up language model pre-training. *arXiv preprint arXiv:2407.06654*, 2024.

[168] Y. He, Z. Wang, Z. Shen, G. Sun, Y. Dai, Y. Wu, H. Wang, and A. Li. Shed: Shapley-based automated dataset refinement for instruction fine-tuning. *arXiv preprint arXiv:2405.00705*, 2024.

[169] D. Hernandez, T. Brown, T. Conerly, N. DasSarma, D. Drain, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, T. Henighan, T. Hume, et al. Scaling laws and interpretability of learning from repeated data. *arXiv preprint arXiv:2205.10487*, 2022.

[170] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. W. Rae, and L. Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[171] S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, C. Zhang, C. Wei, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, X. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Z. Gou, Z. Xu, and C. Wu. Data interpreter: An llm agent for data science, 2024.

[172] M. J. Hosseini, Y. Gao, T. Baumgärtner, A. Fabrikant, and R. K. Amplayo. Scalable and domain-general abstractive proposition segmentation. *arXiv preprint arXiv:2406.19803*, 2024.

[173] Z. Hou, X. Lv, R. Lu, J. Zhang, Y. Li, Z. Yao, J. Li, J. Tang, and Y. Dong. Advancing language model reasoning through reinforcement learning and inference scaling. *arXiv preprint arXiv:2501.11651*, 2025.

[174] A. Hu, H. Xu, J. Ye, M. Yan, L. Zhang, B. Zhang, C. Li, J. Zhang, Q. Jin, F. Huang, and J. Zhou. mplug-docowl 1.5: Unified structure learning for ocr-free document understanding, 2024.

[175] H. Hua, Y. Tang, C. Xu, and J. Luo. V2xum-llm: Cross-modal video summarization with temporal prompt instruction tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 3599–3607, 2025.

[176] J. Huang, D. Guo, C. Wang, J. Gu, S. Lu, J. P. Inala, C. Yan, J. Gao, N. Duan, and M. R. Lyu. Contextualized data-wrangling code generation in computational notebooks. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 1282–1294. ACM, Oct. 2024.

[177] X. Huang, H. Li, J. Zhang, X. Zhao, Z. Yao, Y. Li, Z. Yu, T. Zhang, H. Chen, and C. Li. E2etune: End-to-end knob tuning via fine-tuned generative language model. *CoRR*, abs/2404.11581, 2025.

[178] Y. Huang, X. Lin, Z. Liu, Q. Cao, H. Xin, H. Wang, Z. Li, L. Song, and X. Liang. Mustard: Mastering uniform synthesis of theorem and proof data. *arXiv preprint arXiv:2402.08957*, 2024.

[179] Y. Huang, X. Liu, Y. Gong, Z. Gou, Y. Shen, N. Duan, and W. Chen. Key-point-driven data synthesis with its enhancement on mathematical reasoning. *arXiv preprint arXiv:2403.02333*, 2024.

[180] Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei. Layoutlmv3: Pre-training for document ai with unified text and image masking, 2022.

[181] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[182] G. Ilharco, M. Wortsman, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt. Openclip, July 2021.

[183] I. Ilyankou, M. Wang, S. Cavazzi, and J. Haworth. Cc-gpx: Extracting high-quality annotated geospatial data from common crawl, 2024.

[184] A. Ilyas, S. M. Park, L. Engstrom, G. Leclerc, and A. Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

[185] K. Islam, M. Z. Zaheer, A. Mahmood, and K. Nandakumar. Diffusemix: Label-preserving data augmentation with diffusion models, 2024.

[186] I. Jang, Z. Yang, Z. Zhang, X. Jin, and M. Chowdhury. Oobleck: Resilient distributed training of large models using pipeline templates. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 382–395, 2023.

[187] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[188] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

[189] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

[190] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

[191] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

[192] J. Jiang, K. Zhou, Z. Dong, K. Ye, W. X. Zhao, and J.-R. Wen. Structgpt: A general framework for large language model to reason over structured data, 2023.

[193] J. Jiang, K. Zhou, W. X. Zhao, and J.-R. Wen. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph, 2023.

[194] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, et al. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[195] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[196] R. L. J. X. Y. C. P. H. C. H. M. D. Z. Jie Tan, Kangfei Zhao and Y. Rong. Can large language models be query optimizer for relational databases? *CoRR*, abs/2502.05562, 2025.

[197] C. Jin, Z. Zhang, X. Jiang, F. Liu, X. Liu, X. Liu, and X. Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

[198] D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

[199] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[200] D. Jung, Q. Liu, T. Huang, B. Zhou, and M. Chen. Familiarity-aware evidence compression for retrieval augmented generation. *arXiv preprint arXiv:2409.12468*, 2024.

[201] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[202] A. Kay. Tesseract: an open-source optical character recognition engine. *Linux J.*, 2007(159):2, July 2007.

[203] M. Kayali, A. Lykov, I. Fountalis, N. Vasiloglou, D. Olteanu, and D. Suciu. CHORUS: foundation models for unified data discovery and exploration. *Proc. VLDB Endow.*, 17(8):2104–2114, 2024.

[204] M. Kayali, F. Wenz, N. Tatbul, and Ç. Demiralp. Mind the data gap: Bridging llms to enterprise data integration. In *Proceedings of the 2025 Conference on Innovative Data Systems Research (CIDR)*, Chaminade, California, 2025. CIDR 2025.

[205] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[206] J. D. M.-W. C. Kenton and L. K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota, 2019.

[207] A. Khan, R. Underwood, C. Siebenschuh, Y. Babuji, A. Ajith, K. Hippe, O. Gokdemir, A. Brace, K. Chard, and I. Foster. Lshbloom: Memory-efficient, extreme-scale document deduplication. *arXiv preprint arXiv:2411.04257*, 2024.

[208] M. A. M. Khan, M. S. Bari, X. L. Do, W. Wang, M. R. Parvez, and S. Joty. xcodeeval: A large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval, 2023.

[209] D. Kim, B. Kim, D. Han, and M. Eibich. Autorag: Automated framework for optimization of retrieval augmented generation pipeline, 2024.

[210] J. Kim and J. Lee. Strategic data ordering: Enhancing large language model performance through curriculum learning. *arXiv preprint arXiv:2405.07490*, 2024.

[211] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. Published as a conference paper at ICLR 2017.

[212] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR, 2023.

[213] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[214] D. Kocetkov, R. Li, L. Ben Allal, J. Li, C. Mou, C. Muñoz Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, and H. de Vries. The stack: 3 tb of permissively licensed source code. *Preprint*, 2022.

[215] K. Kolluru, M. Mohammed, S. Mittal, S. Chakrabarti, et al. Alignment-augmented consistent translation for multilingual open information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2502–2517, 2022.

[216] W. Kong, Q. Tian, Z. Zhang, R. Min, Z. Dai, J. Zhou, J. Xiong, X. Li, B. Wu, J. Zhang, et al. Hunyuanvideo: A systematic framework for large video generative models. *arXiv preprint arXiv:2412.03603*, 2024.

[217] K. Korini and C. Bizer. Evaluating knowledge generation and self-refinement strategies for llm-based column type annotation. *CoRR*, abs/2503.02718, 2025.

[218] M. M. Krell, M. Kosec, S. P. Perez, and A. Fitzgibbon. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *arXiv preprint arXiv:2107.02027*, 2021.

[219] A. V. Kumar and M. Sivathanu. Quiver: An informed storage cache for deep learning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 283–296, 2020.

[220] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[221] J. Lai, W. Gan, J. Wu, Z. Qi, and P. S. Yu. Large language models in law: A survey. *AI Open*, 2024.

[222] Z. Lai, H. Zhang, B. Zhang, W. Wu, H. Bai, A. Timofeev, X. Du, Z. Gan, J. Shan, C.-N. Chuah, Y. Yang, and M. Cao. Veclip: Improving clip training via visual-enriched captions, 2024.

[223] J. Lao, Y. Wang, Y. Li, J. Wang, Y. Zhang, Z. Cheng, W. Chen, M. Tang, and J. Wang. Gptuner: A manual-reading database tuning system via gpt-guided bayesian optimization. *Proc. VLDB Endow.*, 17(8):1939–1952, 2024.

[224] H. Laurençon, L. Saulnier, L. Tronchon, S. Bekman, A. Singh, A. Lozhkov, T. Wang, S. Karamcheti, A. Rush, D. Kiela, et al. Obelics: An open web-scale filtered dataset of interleaved image-text documents. *Advances in Neural Information Processing Systems*, 36, 2024.

[225] K. Lee, M. Joshi, I. Turc, H. Hu, F. Liu, J. Eisenschlos, U. Khandelwal, P. Shaw, M.-W. Chang, and K. Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding, 2023.

[226] F. Lei, X. Li, Y. Wei, S. He, Y. Huang, J. Zhao, and K. Liu. S3HQA: A three-stage approach for multi-hop text-table hybrid question answering. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1731–1740, Toronto, Canada, July 2023. Association for Computational Linguistics.

[227] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 2023.

[228] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[229] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang. The dawn of natural language to sql: Are we fully ready? *Proceedings of the VLDB Endowment*, 17(11):3318–3331, July 2024.

[230] D. Li, S. Cao, T. Griggs, S. Liu, X. Mo, S. G. Patil, M. Zaharia, J. E. Gonzalez, and I. Stoica. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025.

[231] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, 12(12):2118–2130, 2019.

[232] H. Li, Y. Chen, Q. Ai, Y. Wu, R. Zhang, and Y. Liu. Lexeval: A comprehensive chinese legal benchmark for evaluating large language models. *arXiv preprint arXiv:2409.20288*, 2024.

[233] H. Li, Q. Dong, Z. Tang, C. Wang, X. Zhang, H. Huang, S. Huang, X. Huang, Z. Huang, D. Zhang, et al. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*, 2024.

[234] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen. Codes: Towards building open-source language models for text-to-sql, 2024.

[235] J. LI, E. Beeching, L. Tunstall, B. Lipkin, R. Soletskyi, S. C. Huang, K. Rasul, L. Yu, A. Jiang, Z. Shen, Z. Qin, B. Dong, L. Zhou, Y. Fleureau, G. Lample, and S. Polu. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

[236] J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*, 2024.

[237] L. Li, L. Fang, and V. I. Torvik. Autodcworkflow: Llm-based data cleaning workflow auto-generation and benchmark. *CoRR*, abs/2412.06724, 2024.

[238] M. Li, Y. Zhang, S. He, Z. Li, H. Zhao, J. Wang, N. Cheng, and T. Zhou. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. *arXiv preprint arXiv:2402.00530*, 2024.

[239] M. Li, Y. Zhang, Z. Li, J. Chen, L. Chen, N. Cheng, J. Wang, T. Zhou, and J. Xiao. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. *arXiv preprint arXiv:2308.12032*, 2023.

[240] P. Li, Y. He, D. Yashar, W. Cui, S. Ge, H. Zhang, D. R. Fainman, D. Zhang, and S. Chaudhuri. Table-gpt: Table-tuned gpt for diverse table tasks, 2023.

[241] S. Li, X. Ning, L. Wang, T. Liu, X. Shi, S. Yan, G. Dai, H. Yang, and Y. Wang. Evaluating quantized large language models. In *ICML*. OpenReview.net, 2024.

[242] X. Li, Z. Wu, J. Wu, H. Cui, J. Jia, R.-H. Li, and G. Wang. Graph learning in the era of llms: A survey from the perspective of data, models, and tasks, 2024.

[243] Y. Li, H. Li, P. Zhao, J. Zhang, X. Zhang, T. Ji, L. Sun, C. Li, and H. Chen. Is large language model good at database knob tuning? A comprehensive experimental evaluation. *CoRR*, abs/2408.02213, 2024.

[244] Y. LI, G. Zhang, X. Qu, J. Li, Z. Li, Z. Wang, H. Li, R. Yuan, Y. Ma, K. Zhang, W. Zhou, Y. Liang, L. Zhang, L. Ma, J. Zhang, Z. Li, S. W. Huang, C. Lin, and J. Fu. Cif-bench: A chinese instruction-following benchmark for evaluating the generalizability of large language models, 2024.

[245] Z. Li, Y. Du, M. Zheng, and M. Song. Mimotable: A multi-scale spreadsheet benchmark with meta operations for table reasoning, 2024.

[246] Z. Li, S. Fan, Y. Gu, X. Li, Z. Duan, B. Dong, N. Liu, and J. Wang. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering, 2024.

[247] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao, and H. Mao. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency, June 2024.

[248] Z. Li, H. Yuan, H. Wang, G. Cong, and L. Bing. LLM-R2: A large language model enhanced rule-based rewrite system for boosting query efficiency. *Proc. VLDB Endow.*, 18(1):53–65, 2024.

[249] Z. Li, X. Zhang, Y. Zhang, D. Long, P. Xie, and M. Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[250] J. Lian, X. Liu, Y. Shao, et al. Chatbi: Towards natural language to complex business intelligence SQL. *CoRR*, abs/2405.00527, 2024.

[251] H. Liang, K. Zhao, Y. Yang, B. Cui, G. Dong, Z. Zhou, and W. Zhang. Data proportion detection for optimized data management for large language models. *arXiv preprint arXiv:2409.17527*, 2024.

[252] Y. Liang, T. Xie, G. Peng, Z. Huang, Y. Lan, and W. Qian. Natnl2gql: A novel multi-agent framework for translating natural language to graph query language, 2024.

[253] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

[254] X. Lin, W. Wang, Y. Li, S. Yang, F. Feng, Y. Wei, and T.-S. Chua. Data-efficient fine-tuning for llm-based recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 365–374, 2024.

[255] A. Liu, J. Liu, Z. Pan, Y. He, R. Haffari, and B. Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

[256] A. Liu, L. Pan, X. Hu, S. Li, L. Wen, I. King, and S. Y. Philip. An unforgeable publicly verifiable watermark for large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

[257] C. Liu, H. Wei, J. Chen, L. Kong, Z. Ge, Z. Zhu, L. Zhao, J. Sun, C. Han, and X. Zhang. Focus anywhere for fine-grained multi-page document understanding, 2024.

[258] H. Liu, C. Li, Y. Li, and Y. J. Lee. Improved baselines with visual instruction tuning, 2024.

[259] H. Liu, Q. Peng, Q. Yang, K. Liu, and H. Xu. Bucket pre-training is all you need. *arXiv preprint arXiv:2407.07495*, 2024.

[260] H. Liu, Y. Zhang, Y. Luo, and A. C.-C. Yao. Augmenting math word problems via iterative question composing. *arXiv preprint arXiv:2401.09003*, 2024.

[261] J. Liu and B. Mozafari. Query rewriting via large language models. *CoRR*, abs/2403.09060, 2024.

[262] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou. Large language model-based agents for software engineering: A survey, 2024.

[263] Q. Liu, X. Zheng, N. Muennighoff, G. Zeng, L. Dou, T. Pang, J. Jiang, and M. Lin. Regmix: Data mixture as regression for language model pre-training. *arXiv preprint arXiv:2407.01492*, 2024.

[264] W. Liu, W. Zeng, K. He, Y. Jiang, and J. He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*, 2023.

[265] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56, 2024.

[266] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[267] Y. Liu, E. Peña, A. S. R. Santos, E. Wu, and J. Freire. Magneto: Combining small and large language models for schema matching. *CoRR*, abs/2412.08194, 2024.

[268] Z. Liu, Y. Huang, X. Yu, L. Zhang, Z. Wu, C. Cao, H. Dai, L. Zhao, Y. Li, P. Shu, et al. Deid-gpt: Zero-shot medical text de-identification by gpt-4. *arXiv preprint arXiv:2303.11032*, 2023.

[269] Z. Liu, A. Karbasi, and T. Rekatsinas. Tsds: Data selection for task-specific model finetuning. *arXiv preprint arXiv:2410.11303*, 2024.

[270] Z. Liu, Q. Liao, W. Gu, and C. Gao. Software vulnerability detection with gpt and in-context learning. In *2023 8th International Conference on Data Science in Cyberspace (DSC)*, pages 229–236, 2023.

[271] Z. Liu, Z. Tang, J. Zhang, X. Xia, and X. Yang. Pre-training by predicting program dependencies for vulnerability analysis tasks, 2024.

[272] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey. *arXiv preprint arXiv:2406.15126*, 2024.

[273] D. Lu, H. Wu, J. Liang, Y. Xu, Q. He, Y. Geng, M. Han, Y. Xin, and Y. Xiao. Bbt-fin: Comprehensive construction of chinese financial domain pre-trained language model, corpus and benchmark. *arXiv preprint arXiv:2302.09432*, 2023.

[274] W. Lu, J. Zhang, J. Fan, Z. Fu, Y. Chen, and X. Du. Large language model for table processing: A survey. *Frontiers of Computer Science*, 19(2):192350, 2025.

[275] N. Lukas, A. Salem, R. Sim, S. Tople, L. Wutschitz, and S. Zanella-Béguelin. Analyzing leakage of personally identifiable information in language models. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 346–363. IEEE, 2023.

[276] Z. Luo, X. Zhang, X. Liu, H. Li, Y. Gong, C. Qi, and P. Cheng. Velocitune: A velocity-based dynamic domain reweighting method for continual pre-training. *arXiv preprint arXiv:2411.14318*, 2024.

[277] C. Ma, S. Chakrabarti, A. Khan, and B. Molnár. Knowledge graph-based retrieval-augmented generation for schema matching. *CoRR*, abs/2501.08686, 2025.

[278] G. Ma, Y. Ma, X. Wu, Z. Su, M. Zhou, and S. Hu. Task-level distributionally robust optimization for large language model-based dense retrieval. *arXiv preprint arXiv:2408.10613*, 2024.

[279] L. Ma, N. Thakurdesai, J. Chen, J. Xu, E. Körpeoglu, S. Kumar, and K. Achan. Llms with user-defined prompts as generic data operators for reliable data processing. In *IEEE Big Data*, pages 3144–3148. IEEE, 2023.

[280] Y. Ma, Y. Cao, Y. Hong, and A. Sun. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023.

[281] Z. Ma, B. Zhang, J. Zhang, J. Yu, X. Zhang, X. Zhang, S. Luo, X. Wang, and J. Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation, 2024.

[282] P. Maini, S. Seto, H. Bai, D. Grangier, Y. Zhang, and N. Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling. *arXiv preprint arXiv:2401.16380*, 2024.

[283] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[284] Y. Mao, X. Li, W. Li, X. Wang, and L. Xie. Scla: Automated smart contract summarization via llms and semantic augmentation, 2024.

[285] M. Marion, A. Üstün, L. Pozzobon, A. Wang, M. Fadaee, and S. Hooker. When less is more: Investigating data pruning for pretraining llms at scale. *arXiv preprint arXiv:2309.04564*, 2023.

[286] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[287] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[288] D. Mekala, A. Nguyen, and J. Shang. Smaller language models are capable of selecting instruction-tuning training data for larger language models. *arXiv preprint arXiv:2402.10430*, 2024.

[289] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

[290] A. Mitra, L. Del Corro, G. Zheng, S. Mahajan, D. Rouhana, A. Codas, Y. Lu, W.-g. Chen, O. Vrousgos, C. Rosset, et al. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*, 2024.

[291] J. Mohan, A. Phanishayee, and V. Chidambaram. CheckFreq: Frequent, Fine-Grained DNN checkpointing. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 203–216. USENIX Association, Feb. 2021.

[292] J. Monteiro, F. Sá, and J. Bernardino. Graph databases assessment: Janusgraph, neo4j, and tigergraph. In *Perspectives and Trends in Education and Technology: Selected Papers from ICITED 2022*, pages 655–665. Springer, 2023.

[293] J. Mu, X. Li, and N. Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36, 2024.

[294] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. von Werra, and S. Longpre. Octopack: Instruction tuning code large language models, 2024.

[295] C. Na, I. Magnusson, A. H. Jha, T. Sherborne, E. Strubell, J. Dodge, and P. Dasigi. Scalable data ablation approximations for language models through modular training and merging. *arXiv preprint arXiv:2410.15661*, 2024.

[296] R. Navigli, S. Conia, and B. Ross. Biases in large language models: origins, inventory, and discussion. *ACM Journal of Data and Information Quality*, 15(2):1–21, 2023.

[297] T. Nguyen, C. V. Nguyen, V. D. Lai, H. Man, N. T. Ngo, F. Dernoncourt, R. A. Rossi, and T. H. Nguyen. CulturaX: A cleaned, enormous, and multilingual dataset for large language models in 167 languages. In N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4226–4237, Torino, Italia, May 2024. ELRA and ICCL.

[298] I. Nunes, M. Heddes, P. Vergés, D. Abraham, A. Veidenbaum, A. Nicolau, and T. Givargis. Dothash: Estimating set similarity metrics for link prediction and document deduplication. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1758–1769, 2023.

[299] A. Nystrom, C. Zhang, C. Callison-Burch, D. Ippolito, D. Eck, K. Lee, and N. Carlini. Deduplicating training data makes language models better. 2022.

[300] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[301] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[302] R. Pan, J. Zhang, X. Pan, R. Pi, X. Wang, and T. Zhang. Scalebio: Scalable bilevel optimization for llm data reweighting. *arXiv preprint arXiv:2406.19976*, 2024.

[303] Z. Pan, Q. Wu, H. Jiang, M. Xia, X. Luo, J. Zhang, Q. Lin, V. Rühle, Y. Yang, C.-Y. Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*, 2024.

[304] M. Parciak, B. Vandevoort, F. Neven, L. M. Peeters, and S. Vansummeren. Schema matching with large language models: an experimental study. In *VLDB Workshops*. VLDB.org, 2024.

[305] H. Park, S. Lee, G. Gim, Y. Kim, D. Kim, and C. Park. Dataverse: Open-source etl (extract, transform, load) pipeline for large language models. *arXiv preprint arXiv:2403.19340*, 2024.

[306] S. Patnaik, H. Changwal, M. Aggarwal, S. Bhatia, Y. Kumar, and B. Krishnamurthy. Cabinet: Content relevance based noise reduction for table question answering, 2024.

[307] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, 1988.

[308] R. Peeters, A. Steiner, and C. Bizer. Entity matching using large language models. In *EDBT*, pages 529–541. OpenProceedings.org, 2025.

[309] Q. Pei, L. Wu, K. Gao, J. Zhu, Y. Wang, Z. Wang, T. Qin, and R. Yan. Leveraging biomolecule and natural language through multi-modal learning: A survey, 2024.

[310] G. Penedo, H. Kydlíček, A. Lozhkov, M. Mitchell, C. Raffel, L. Von Werra, T. Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

[311] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

[312] B. Peng, C. Li, P. He, M. Galley, and J. Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

[313] M. E. Peters and D. Lecocq. Content extraction using diverse feature sets. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 89–90, New York, NY, USA, 2013. Association for Computing Machinery.

[314] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023.

[315] J. Postel. Transmission control protocol. Technical report, 1981.

[316] H. Pouransari, C.-L. Li, J.-H. R. Chang, P. K. A. Vasu, C. Koc, V. Shankar, and O. Tuzel. Dataset decomposition: Faster llm training with variable sequence length curriculum. *arXiv preprint arXiv:2405.13226*, 2024.

[317] M. Pourreza and D. Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction, 2023.

[318] R. Pradeep, S. Sharifymoghaddam, and J. Lin. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088*, 2023.

[319] D. Qi and J. Wang. Cleanagent: Automating data standardization with llm-based agents. *CoRR*, abs/2403.08291, 2024.

[320] Z. Qiang, W. Wang, and K. Taylor. Agent-om: Leveraging llm agents for ontology matching. *arXiv preprint arXiv:2312.00326*, 2023.

[321] R. Qin, J. Xia, Z. Jia, M. Jiang, A. Abbasi, P. Zhou, J. Hu, and Y. Shi. Enabling on-device large language model personalization with self-supervised data selection and synthesis. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[322] Z. Qin, D. Chen, W. Zhang, L. Yao, Y. Huang, B. Ding, Y. Li, and S. Deng. The synergy between data and multi-modal large language models: A survey from co-development perspective. *arXiv preprint arXiv:2407.08583*, 2024.

[323] H. Que, J. Liu, G. Zhang, C. Zhang, X. Qu, Y. Ma, F. Duan, Z. Bai, J. Wang, Y. Zhang, et al. D-cpt law: Domain-specific continual pre-training scaling law for large language models. *arXiv preprint arXiv:2406.01375*, 2024.

[324] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report, 2025.

[325] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[326] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

[327] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[328] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[329] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.

[330] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[331] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[332] R. Rahnamoun and M. Shamsfard. Multi-layered evaluation using a fusion of metrics and llms as judges in open-domain question answering. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6088–6104, 2025.

[333] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[334] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–14, 2021.

[335] D. Rau, S. Wang, H. Déjean, and S. Clinchant. Context embeddings for efficient answer generation in rag. *arXiv preprint arXiv:2407.09252*, 2024.

[336] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He. Zero-offload: Democratizing billion-scale model training, 2021.

[337] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[338] S. Robertson and H. Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, Apr. 2009.

[339] A. E. Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[340] A. S. R. Santos, E. H. M. Pena, R. Lopez, and J. Freire. Interactive data harmonization with LLM agents. *CoRR*, abs/2502.07132, 2025.

[341] C. Schuhmann, R. Vencu, R. Beaumont, R. Kaczmarczyk, C. Mullis, A. Katta, T. Coombes, J. Jitsev, and A. Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*, 2021.

[342] O. Sener and S. Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[343] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[344] Y. Shao, L. Li, Y. Ma, P. Li, D. Song, Q. Cheng, S. Li, X. Li, P. Wang, Q. Guo, et al. Case2code: Scalable synthetic data for code generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 11056–11069, 2025.

[345] H. Shen, P.-Y. Chen, P. Das, and T. Chen. Seal: Safety-enhanced aligned llm fine-tuning via bilevel data selection. *arXiv preprint arXiv:2410.07471*, 2024.

[346] M. Shen, G. Zeng, Z. Qi, Z.-W. Hong, Z. Chen, W. Lu, G. Wornell, S. Das, D. Cox, and C. Gan. Satori: Reinforcement learning with chain-of-action-thought enhances llm reasoning via autoregressive search. *arXiv preprint arXiv:2502.02508*, 2025.

[347] Z. Shen, T. Tao, L. Ma, W. Neiswanger, Z. Liu, H. Wang, B. Tan, J. Hestness, N. Vassilieva, D. Soboleva, et al. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*, 2023.

[348] K. Shi, X. Sun, Q. Li, and G. Xu. Compressing long context for enhancing rag with amr-based concept distillation. *arXiv preprint arXiv:2405.03085*, 2024.

[349] W. Shi, S. Min, M. Lomeli, C. Zhou, M. Li, G. Szilvasy, R. James, X. V. Lin, N. A. Smith, L. Zettlemoyer, et al. In-context pretraining: Language modeling beyond document boundaries. *arXiv preprint arXiv:2310.10638*, 2023.

[350] Y. Shi, X. Zi, Z. Shi, H. Zhang, Q. Wu, and M. Xu. Eragent: Enhancing retrieval-augmented language models with improved accuracy, efficiency, and personalization. *arXiv preprint arXiv:2405.06683*, 2024.

[351] Z. Shi, S. Gao, L. Yan, Y. Feng, X. Chen, Z. Chen, D. Yin, S. Verberne, and Z. Ren. Tool learning in the wild: Empowering language models as automatic tool agents. In *Proceedings of the ACM on Web Conference 2025*, pages 2222–2237, 2025.

[352] L. Shimabucoro, S. Ruder, J. Kreutzer, M. Fadaee, and S. Hooker. Llm see, llm do: Guiding data generation to target non-differentiable objectives (2024). *URL https://arxiv.org/abs/2407.01490*.

[353] A. Shirgaonkar, N. Pandey, N. C. Abay, T. Aktas, and V. Aski. Knowledge distillation using frontier open-source llms: Generalizability and the role of synthetic data. *arXiv preprint arXiv:2410.18588*, 2024.

[354] K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[355] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[356] A. Shrivastava and P. Li. In defense of minhash over simhash. In *Artificial Intelligence and Statistics*, pages 886–894. PMLR, 2014.

[357] D. Shrivastava, D. Kocetkov, H. de Vries, D. Bahdanau, and T. Scholak. RepoFusion: Training Code Models to Understand Your Repository, June 2023.

[358] E. Silcock, L. D'Amico-Wong, J. Yang, and M. Dell. Noise-robust de-duplication at scale. Technical report, National Bureau of Economic Research, 2022.

[359] V. Y. Singh, K. Vaidya, V. B. Kumar, S. Khosla, B. Narayanaswamy, R. Gangadharaiah, and T. Kraska. Panda: Performance debugging for databases using LLM agents. In *CIDR*. www.cidrdb.org, 2024.

[360] E. Slyman, S. Lee, S. Cohen, and K. Kafle. Fairdedup: Detecting and mitigating vision-language fairness disparities in semantic dataset deduplication. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13905–13916, 2024.

[361] L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, et al. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*, 2024.

[362] J. Song, Z. Zhang, Z. Tang, S. Feng, and Y. Gu. Improving code summarization with tree transformer enhanced by position-related syntax complement. *IEEE Transactions on Artificial Intelligence*, 5:4776–4786, 2024.

[363] J. H. Sriram Dharwada, Himanshu Devrani and H. Doraiswamy. Query rewriting via llms. *CoRR*, abs/2502.12918, 2025.

[364] K. Staniszewski, S. Tworkowski, S. Jaszczur, Y. Zhao, H. Michalewski, Ł. Kuciński, and P. Miłoś. Structured packing in llm training improves long context utilization. *arXiv preprint arXiv:2312.17296*, 2023.

[365] A. Su, A. Wang, C. Ye, C. Zhou, G. Zhang, G. Chen, G. Zhu, H. Wang, H. Xu, H. Chen, H. Li, H. Lan, J. Tian, J. Yuan, J. Zhao, J. Zhou, K. Shou, L. Zha, L. Long, L. Li, P. Wu, Q. Zhang, Q. Huang, S. Yang, T. Zhang, W. Ye, W. Zhu, X. Hu, X. Gu, X. Sun, X. Li, Y. Yang, and Z. Xiao. Tablegpt2: A large multimodal model with tabular data integration, 2024.

[366] S. Sudalairaj, A. Bhandwaldar, A. Pareja, K. Xu, D. D. Cox, and A. Srivastava. Lab: Large-scale alignment for chatbots. *arXiv preprint arXiv:2403.01081*, 2024.

[367] L. Sun, K. Zhang, Q. Li, and R. Lou. Umie: Unified multimodal information extraction with instruction tuning, 2024.

[368] Y. Sun, F. Wang, Y. Zhu, W. X. Zhao, and J. Mao. An integrated data processing framework for pretraining foundation models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2713–2718, 2024.

[369] Z. Sun, X. Zhou, and G. Li. R-bot: An llm-based query rewrite system. *CoRR*, abs/2412.01661, 2024.

[370] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi. Chess: Contextual harnessing for efficient sql synthesis, 2024.

[371] A. Talmor, J. Herzig, N. Lourie, and J. Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

[372] H. Tan, S. Wu, F. Du, Y. Chen, Z. Wang, F. Wang, and X. Qi. Data pruning via moving-one-sample-out. *Advances in Neural Information Processing Systems*, 36, 2024.

[373] Z. Tan, D. Li, S. Wang, A. Beigi, B. Jiang, A. Bhattacharjee, M. Karami, J. Li, L. Cheng, and H. Liu. Large language models for data annotation and synthesis: A survey. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. arXiv preprint arXiv:2402.13446.

[374] Z. Tan, D. Li, S. Wang, et al. Large language models for data annotation and synthesis: A survey. In *EMNLP*, pages 930–957. Association for Computational Linguistics, 2024.

[375] J. Tang, Y. Yang, W. Wei, L. Shi, L. Su, S. Cheng, D. Yin, and C. Huang. Graphgpt: Graph instruction tuning for large language models, 2024.

[376] Z. Tang, Z. Yang, G. Wang, Y. Fang, Y. Liu, C. Zhu, M. Zeng, C. Zhang, and M. Bansal. Unifying vision, text, and layout for universal document processing, 2023.

[377] K. Team, A. Du, B. Gao, B. Xing, et al. Kimi k1.5: Scaling reinforcement learning with llms, 2025.

[378] M. N. Team et al. Introducing mpt-7b: A new standard for open-source, commercially usable llms. *DataBricks (May, 2023) www. mosaicml. com/blog/mpt-7b*, 2023.

[379] Q. Team. Qwq: Reflect deeply on the boundaries of the unknown. *Hugging Face*, 2024.

[380] M. Tepper, I. S. Bhati, C. Aguerrebere, M. Hildebrand, and T. Willke. Leanvec: Searching vectors faster by making them fit, 2024.

[381] M. Tepper, I. S. Bhati, C. Aguerrebere, and T. Willke. Gleanvec: Accelerating vector search with minimalist nonlinear dimensionality reduction, 2024.

[382] J. Thorpe, P. Zhao, J. Eyolfson, Y. Qiao, Z. Jia, M. Zhang, R. Netravali, and G. H. Xu. Bamboo: Making preemptible instances resilient for affordable training of large {DNNs}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 497–513, 2023.

[383] T. Thrush, C. Potts, and T. Hashimoto. Improving pretraining data using perplexity correlations. *arXiv preprint arXiv:2409.05816*, 2024.

[384] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes. Borg: the next generation. In *Proceedings of the fifteenth European conference on computer systems*, pages 1–14, 2020.

[385] K. Tirumala, D. Simig, A. Aghajanyan, and A. Morcos. D4: Improving llm pretraining via document de-duplication and diversification. *Advances in Neural Information Processing Systems*, 36:53983–53995, 2023.

[386] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[387] B. Trabucco, K. Doherty, M. Gurinas, and R. Salakhutdinov. Effective data augmentation with diffusion models, 2023.
[388] G. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.
[389] B. Wan, M. Han, Y. Sheng, Y. Peng, H. Lin, M. Zhang, Z. Lai, M. Yu, J. Zhang, Z. Song, X. Liu, and C. Wu. Bytecheckpoint: A unified checkpointing system for large foundation model development, 2024.
[390] A. Wang, B. Ai, B. Wen, C. Mao, C.-W. Xie, D. Chen, F. Yu, H. Zhao, J. Yang, J. Zeng, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.
[391] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding. Yolov10: Real-time end-to-end object detection, 2024.
[392] B. Wang, C. Xu, X. Zhao, L. Ouyang, F. Wu, Z. Zhao, R. Xu, K. Liu, Y. Qu, F. Shang, B. Zhang, L. Wei, Z. Sui, W. Li, B. Shi, Y. Qiao, D. Lin, and C. He. Mineru: An open-source solution for precise document content extraction, 2024.
[393] C. Wang, Q. Dong, X. Wang, H. Wang, and Z. Sui. Statistical dataset evaluation: Reliability, difficulty, and validity, 2022.
[394] C. Wang, M. Li, J. He, Z. Wang, E. Darzi, Z. Chen, J. Ye, T. Li, Y. Su, J. Ke, et al. A survey for large language models in biomedicine. *arXiv preprint arXiv:2409.00133*, 2024.
[395] C. Wang, Q. Wu, S. Huang, and A. Saied. Economic hyperparameter optimization with blended search strategy. In *International Conference on Learning Representations*, 2021.
[396] H. Wang, J. Wang, C. T. Leong, and W. Li. Steca: Step-level trajectory calibration for llm agent learning, 2025.
[397] J. Wang, J. Wu, Y. Hou, Y. Liu, M. Gao, and J. McAuley. Instructgraph: Boosting large language models via graph-centric instruction tuning and preference alignment, 2024.
[398] J. Wang, B. Zhang, Q. Du, J. Zhang, and D. Chu. A survey on data selection for llm instruction tuning. *arXiv preprint arXiv:2402.05123*, 2024.
[399] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.
[400] T. Wang, X. Chen, H. Lin, X. Chen, X. Han, L. Sun, H. Wang, and Z. Zeng. Match, compare, or select? an investigation of large language models for entity matching. In *COLING*, pages 96–109. Association for Computational Linguistics, 2025.
[401] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.
[402] Z. Wang, X. He, K. Chen, C. Lin, and J. Su. Code-aware cross-program transfer hyperparameter optimization. In *AAAI*, pages 10297–10305. AAAI Press, 2023.
[403] Z. Wang, Z. Jia, S. Zheng, Z. Zhang, X. Fu, T. E. Ng, and Y. Wang. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 364–381, 2023.
[404] Z. Wang, H. Zhang, C.-L. Li, J. M. Eisenschlos, V. Perot, Z. Wang, L. Miculicich, Y. Fujii, J. Shang, C.-Y. Lee, and T. Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding, 2024.
[405] Z. Wang, W. Zhong, Y. Wang, Q. Zhu, F. Mi, B. Wang, L. Shang, X. Jiang, and Q. Liu. Data management for training large language models: A survey. *arXiv preprint arXiv:2312.01700*, 2023.
[406] H. Wei, L. Kong, J. Chen, L. Zhao, Z. Ge, J. Yang, J. Sun, C. Han, and X. Zhang. Vary: Scaling up the vision vocabulary for large vision-language models, 2023.
[407] H. Wei, C. Liu, J. Chen, J. Wang, L. Kong, Y. Xu, Z. Ge, L. Zhao, J. Sun, Y. Peng, C. Han, and X. Zhang. General ocr theory: Towards ocr-2.0 via a unified end-to-end model, 2024.
[408] L. Wei, G. Xiao, and M. Balazinska. RACOON: an llm-based framework for retrieval-augmented column type annotation with a knowledge graph. *CoRR*, abs/2409.14556, 2024.
[409] Y. Wei, Z. Wang, J. Liu, Y. Ding, and L. Zhang. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*, 10, 2023.
[410] G. Wenzek, M.-A. Lachaux, A. Conneau, V. Chaudhary, F. Guzmán, A. Joulin, and E. Grave. Ccnet: Extracting high quality monolingual datasets from web crawl data. *arXiv preprint arXiv:1911.00359*, 2019.
[411] A. Wettig, A. Gupta, S. Malik, and D. Chen. Qurating: Selecting high-quality data for training language models. *arXiv preprint arXiv:2402.09739*, 2024.
[412] C. Whitehouse, C. Vania, A. F. Aji, C. Christodoulopoulos, and A. Pierleoni. Webie: Faithful and robust information extraction on the web. *arXiv preprint arXiv:2305.14293*, 2023.
[413] D. Wu, W. U. Ahmad, D. Zhang, M. K. Ramanathan, and X. Ma. Repoformer: Selective Retrieval for Repository-Level Code Completion, June 2024.
[414] H. Wu, E. Zhang, L. Liao, C. Chen, J. Hou, A. Wang, W. Sun, Q. Yan, and W. Lin. Exploring video quality assessment on user generated contents from aesthetic and technical perspectives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20144–20154, 2023.
[415] J. Wu, J. Zhu, and Y. Qi. Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation. *arXiv preprint arXiv:2408.04187*, 2024.
[416] M. Wu, T.-T. Vu, L. Qu, and G. Haffari. Mixture-of-skills: Learning to optimize data usage for fine-tuning large language models. *arXiv preprint arXiv:2406.08811*, 2024.
[417] M. Xia, S. Malladi, S. Gururangan, S. Arora, and D. Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.
[418] B. Xiao, H. Wu, W. Xu, X. Dai, H. Hu, Y. Lu, M. Zeng, C. Liu, and L. Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks, 2023.
[419] C. Xie, Z. Lin, A. Backurs, S. Gopi, D. Yu, H. A. Inan, H. Nori, H. Jiang, H. Zhang, Y. T. Lee, et al. Differentially private synthetic data via foundation model apis 2: Text. *arXiv preprint arXiv:2403.01749*, 2024.
[420] S. M. Xie, H. Pham, X. Dong, N. Du, H. Liu, Y. Lu, P. S. Liang, Q. V. Le, T. Ma, and A. W. Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.
[421] S. M. Xie, S. Santurkar, T. Ma, and P. S. Liang. Data selection for language models via importance resampling. *Advances in Neural Information Processing Systems*, 36:34201–34227, 2023.
[422] W. Xie. Analysis of the reasoning with redundant information provided ability of large language models. *arXiv preprint arXiv:2310.04039*, 2023.
[423] Y. Xie, K. Aggarwal, and A. Ahmad. Efficient continual pre-training for building domain specific large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 10184–10201, 2024.
[424] G. Xiong, J. Bao, and W. Zhao. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models, 2024.
[425] H. Xiu, L. Zhang, T. Zhang, J. Yang, and J. Chen. Query performance explanation through large language model for HTAP systems. *CoRR*, abs/2412.01709, 2024.
[426] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.
[427] F. Xu, W. Shi, and E. Choi. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408*, 2023.
[428] J. Xu, R. Zhang, C. Guo, W. Hu, Z. Liu, F. Wu, Y. Feng, S. Sun, C. Shao, Y. Guo, J. Zhao, K. Zhang, M. Guo, and J. Leng. vtensor: Flexible virtual tensor management for efficient llm serving, 2024.
[429] M. Xu. Medicalgpt: Training medical gpt model. https://github.com/shibing624/MedicalGPT, 2023.
[430] Y. Xu, H. Li, K. Chen, and L. Shou. Kcmf: A knowledge-compliant framework for schema and entity matching with fine-tuning-free llms. *CoRR*, abs/2410.12480, 2024.
[431] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mt5: A massively multilingual pre-trained text-to-text transformer, 2021.
[432] M. Yan, Y. Wang, Y. Wang, X. Miao, and J. Li. GIDCL: A graph-enhanced interpretable data cleaning framework with large language models. *Proc. ACM Manag. Data*, 2(6):236:1–236:29, 2024.
[433] A. Yang, B. Xiao, B. Wang, B. Zhang, C. Bian, C. Yin, C. Lv, D. Pan, D. Wang, D. Yan, et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

[434] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan. Qwen2 technical report, 2024.

[435] H. Yang, J. Zhou, Y. Fu, X. Wang, R. Roane, H. Guan, and T. Liu. Protrain: Efficient llm training via memory-aware techniques. *arXiv preprint arXiv:2406.08334*, 2024.

[436] Y. Yang, S. Mishra, J. Chiang, and B. Mirzasoleiman. Smalltolarge (s2l): Scalable data selection for fine-tuning large language models by summarizing training trajectories of small models. *Advances in Neural Information Processing Systems*, 37:83465–83496, 2024.

[437] Z. Yang, J. Teng, W. Zheng, M. Ding, S. Huang, J. Xu, Y. Yang, W. Hong, X. Zhang, G. Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

[438] Z. Yao, H. Li, J. Zhang, C. Li, and H. Chen. A query optimization method utilizing large language models. *CoRR*, abs/2503.06902, 2025.

[439] J. Ye, P. Liu, T. Sun, Y. Zhou, J. Zhan, and X. Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. *arXiv preprint arXiv:2403.16952*, 2024.

[440] L. Ye, Z. Tao, Y. Huang, and Y. Li. Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition. *arXiv preprint arXiv:2402.15220*, 2024.

[441] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang. Language is all a graph needs, 2024.

[442] Y. Ye, Z. Huang, Y. Xiao, E. Chern, S. Xia, and P. Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

[443] P. Yin, W.-D. Li, K. Xiao, A. Rao, Y. Wen, K. Shi, J. Howland, P. Bailey, M. Catasta, H. Michalewski, A. Polozov, and C. Sutton. Natural language to code generation in interactive data science notebooks, 2022.

[444] S. Yin, C. Fu, S. Zhao, K. Li, X. Sun, T. Xu, and E. Chen. A survey on multimodal large language models. *CoRR*, abs/2306.13549, 2023.

[445] S. Yokoo. Contrastive learning with large memory bank and negative embedding subtraction for accurate copy detection. *arXiv preprint arXiv:2112.04323*, 2021.

[446] L. Yuan, G. Cui, H. Wang, N. Ding, X. Wang, J. Deng, B. Shan, H. Chen, R. Xie, Y. Lin, Z. Liu, B. Zhou, H. Peng, Z. Liu, and M. Sun. Advancing llm reasoning generalists with preference trees, 2024.

[447] S. Yue, W. Chen, S. Wang, B. Li, C. Shen, S. Liu, Y. Zhou, Y. Xiao, S. Yun, X. Huang, et al. Disc-lawllm: Fine-tuning large language models for intelligent legal services. *arXiv preprint arXiv:2309.11325*, 2023.

[448] X. Yue, Y. Ni, K. Zhang, T. Zheng, R. Liu, G. Zhang, S. Stevens, D. Jiang, W. Ren, Y. Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567, 2024.

[449] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[450] S. Zeng, J. Zhang, P. He, J. Ren, T. Zheng, H. Lu, H. Xu, H. Liu, Y. Xing, and J. Tang. Mitigating the privacy issues in retrieval-augmented generation (rag) via pure synthetic data. *arXiv preprint arXiv:2406.14773*, 2024.

[451] S. Zerhoudi and M. Granitzer. Personarag: Enhancing retrieval-augmented generation systems with user-centric agents. In *IR-RAG@SIGIR*, volume 3784 of *CEUR Workshop Proceedings*, pages 1–11. CEUR-WS.org, 2024.

[452] C. Zhang, Y. Mao, Y. Fan, Y. Mi, Y. Gao, L. Chen, D. Lou, and J. Lin. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis, 2024.

[453] F. Zhang, D. Zhu, J. Ming, Y. Jin, D. Chai, L. Yang, H. Tian, Z. Fan, and K. Chen. DH-RAG: A dynamic historical context-powered retrieval-augmented generation method for multi-turn dialogue. *CoRR*, abs/2502.13847, 2025.

[454] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada. Jellyfish: A large language model for data preprocessing. *CoRR*, abs/2312.01678, 2023.

[455] H. Zhang, X. Li, and L. Bing. Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*, 2023.

[456] H. Zhang, Y. Liu, W. Hung, A. S. R. Santos, and J. Freire. Autoddg: Automated dataset description generation using large language models. *CoRR*, abs/2502.01050, 2025.

[457] J. Zhang, Z. Liu, X. Hu, X. Xia, and S. Li. Vulnerability detection by learning from syntax-based execution paths of code. *IEEE Transactions on Software Engineering*, 49(8):4196–4212, 2023.

[458] J. Zhang, X. Zhang, J. Yu, J. Tang, J. Tang, C. Li, and H. Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[459] P. Zhang, G. Zeng, T. Wang, and W. Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[460] S. Zhang, L. Dong, X. Li, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

[461] S. Zhang, Z. Huang, and E. Wu. Data cleaning using large language models. *CoRR*, abs/2410.15547, 2024.

[462] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[463] X. Zhang, H. Wu, Y. Li, Z. Tang, J. Tan, F. Li, and B. Cui. An efficient transfer learning based configuration adviser for database tuning. *Proc. VLDB Endow.*, 17(3):539–552, 2023.

[464] Y. Zhang, J. Henkel, A. Floratou, J. Cahoon, S. Deep, and J. M. Patel. Reactable: Enhancing react for table question answering, 2023.

[465] Y. Zhang, Y. Luo, Y. Yuan, and A. C. Yao. Autonomous data selection with language models for mathematical texts. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2024.

[466] H. Zhao, Z. Han, Z. Yang, Q. Zhang, M. Li, F. Yang, Q. Zhang, B. Li, Y. Yang, L. Qiu, et al. Silod: A co-design of caching and scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 883–898, 2023.

[467] J. Zhao, W. Zhao, A. Drozdov, B. Rozonoyer, M. A. Sultan, J.-Y. Lee, M. Iyyer, and A. McCallum. Multistage collaborative knowledge distillation from a large language model for semi-supervised sequence generation. *arXiv preprint arXiv:2311.08640*, 2023.

[468] M. Zhao, E. Adamiak, and C. Kozyrakis. cedar: Optimized and unified machine learning input data pipelines. *arXiv preprint arXiv:2401.08895*, 2024.

[469] M. Zhao, S. Pan, N. Agarwal, Z. Wen, D. Xu, A. Natarajan, P. Kumar, S. S. P, R. Tijoriwala, K. Asher, H. Wu, A. Basant, D. Ford, D. David, N. Yigitbasi, P. Singh, C.-J. Wu, and C. Kozyrakis. Tectonic-Shift: A composite storage fabric for Large-Scale ML training. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 433–449, Boston, MA, July 2023. USENIX Association.

[470] R. Zhao, Z. L. Thai, Y. Zhang, S. Hu, Y. Ba, J. Zhou, J. Cai, Z. Liu, and M. Sun. Decoratelm: Data engineering through corpus rating, tagging, and editing with language models. *arXiv preprint arXiv:2410.05639*, 2024.

[471] W. Zhao, H. Feng, Q. Liu, J. Tang, S. Wei, B. Wu, L. Liao, Y. Ye, H. Liu, W. Zhou, H. Li, and C. Huang. Tabpedia: Towards comprehensive visual table understanding with concept synergy, 2024.

[472] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[473] X. Zhao, H. Li, J. Zhang, X. Huang, T. Zhang, J. Chen, R. Shi, C. Li, and H. Chen. Llmidxadvis: Resource-efficient index advisor utilizing large language model. *CoRR*, abs/2503.07884, 2025.

[474] X. Zhao, X. Zhou, and G. Li. Automatic database knob tuning: A survey. *IEEE Trans. Knowl. Data Eng.*, 35(12):12470–12490, 2023.

[475] Y. Zhao, L. Chen, A. Cohan, and C. Zhao. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[476] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[477] M. Zheng, X. Feng, Q. Si, Q. She, Z. Lin, W. Jiang, and W. Wang. Multimodal table understanding, 2024.

[478] Z. Zheng, X. Ji, T. Fang, F. Zhou, C. Liu, and G. Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching. *arXiv preprint arXiv:2412.03594*, 2024.

[479] Y. Zhong, Z. Zhang, B. Wu, S. Liu, Y. Chen, C. Wan, H. Hu, L. Xia, R. Ming, Y. Zhu, and X. Jin. Optimizing RLHF training for large language models with stage fusion. In *NSDI*, pages 489–503. USENIX Association, 2025.

[480] Z. Zhong, H. Liu, X. Cui, X. Zhang, and Z. Qin. Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. *arXiv preprint arXiv:2406.00456*, 2024.

[481] K. Zhou, B. Zhang, J. Wang, Z. Chen, W. X. Zhao, J. Sha, Z. Sheng, S. Wang, and J.-R. Wen. Jiuzhang3. 0: Efficiently improving mathematical reasoning by training small data synthesis models. *arXiv preprint arXiv:2405.14365*, 2024.

[482] T. Zhou, X. Zhao, X. Xu, and S. Ren. Bileve: Securing text provenance in large language models against spoofing with bi-level signature. *arXiv preprint arXiv:2406.01946*, 2024.

[483] W. Zhou, Y. Gao, X. Zhou, and G. Li. Cracking SQL Barriers: An llm-based dialect transaltion system. *Proc. ACM Manag. Data*, 3(3 (SIGMOD)), 2025.

[484] W. Zhou, Y. Gao, X. Zhou, and G. Li. Cracksql: A hybrid sql dialect translation system powered by large language models. *arXiv Preprint*, 2025.

[485] W. Zhou, C. Lin, X. Zhou, and G. Li. Breaking it down: An in-depth study of index advisors. *Proc. VLDB Endow.*, 17(10):2405–2418, 2024.

[486] W. Zhou, C. Lin, X. Zhou, G. Li, and T. Wang. Demonstration of vita: Visualizing, testing and analyzing index advisors. In *CIKM*, pages 5133–5137. ACM, 2023.

[487] W. Zhou, C. Lin, X. Zhou, G. Li, and T. Wang. TRAP: tailored robustness assessment for index advisors via adversarial perturbation. In *ICDE*, pages 42–55. IEEE, 2024.

[488] X. Zhou, C. Chai, G. Li, and J. Sun. Database meets artificial intelligence: A survey. *TKDE*, 2020.

[489] X. Zhou, G. Li, and Z. Liu. LLM as DBA. *CoRR*, abs/2308.05481, 2023.

[490] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, J. Wu, J. Liu, R. Feng, and G. Zeng. D-bot: Database diagnosis system using large language models. *Proc. VLDB Endow.*, 17(10):2514–2527, 2024.

[491] X. Zhou, Z. Sun, and G. Li. DB-GPT: large language model meets database. *Data Sci. Eng.*, 9(1):102–111, 2024.

[492] X. Zhou, T. Zhang, and D. Lo. Large language model for vulnerability detection: Emerging results and future directions, 2024.

[493] Y. Zhou, Y. He, S. Tian, Y. Ni, Z. Yin, X. Liu, C. Ji, S. Liu, X. Qiu, G. Ye, and H. Chai. r^3-NL2GQL: A model coordination and knowledge graph alignment approach for NL2GQL. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 13679–13692, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics.

[494] F. Zhu, Z. Liu, F. Feng, C. Wang, M. Li, and T. S. Chua. Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, page 310–318, New York, NY, USA, 2024. Association for Computing Machinery.

[495] X. Zhu, D. Cheng, H. Li, K. Zhang, E. Hua, X. Lv, N. Ding, Z. Lin, Z. Zheng, and B. Zhou. How to synthesize text data without model collapse? *arXiv preprint arXiv:2412.14689*, 2024.

[496] X. Zhu, B. Qi, K. Zhang, X. Long, Z. Lin, and B. Zhou. Pad: Program-aided distillation can teach small models reasoning better than chain-of-thought fine-tuning, 2024.

[497] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.