Data Management for Machine Learning: A Survey

Chengliang Chai Jiayi Wang Yuyu Luo Zeping Niu Guoliang Li

Abstract—Machine learning (ML) has widespread applications and has revolutionized many industries, but suffers from several challenges. First, sufficient high-quality training data is inevitable for producing a well-performed model, but the data is always human expensive to acquire. Second, a large amount of training data and complicated model structures lead to the inefficiency of training and inference. Third, given an ML task, one always needs to train lots of models, which are hard to manage in real applications. Fortunately, database techniques can benefit ML by addressing the above three challenges. In this paper, we review existing studies from the following three aspects along with the pipeline highly related to ML. (1) Data preparation (Pre-ML): it focuses on preparing high-quality training data that can improve the performance of the ML model, where we review data discovery, data cleaning and data labeling. (2) Model training & inference (In-ML): researchers in ML community focus on improving the model performance during training, while in this survey we mainly study how to accelerate the entire training process, also including feature selection and model selection. (3) Model management (Post-ML): in this part, we survey how to store, query, deploy and debug the models after training. Finally, we provide research challenges and future directions.

Index Terms—Database, Machine learning, Data preparation, Model training, Model inference

1 INTRODUCTION

Machine learning (ML) is gaining much popularity due to its power and mystery in terms of accuracy and generalization ability, which has widespread applications, e.g., image recognition [71], natural language processing [56], [99], [102], [103], etc. In the ML community, researchers mainly study how to design sophisticated model structures for higher performance. However, people always suffer from several challenges when they utilize ML in real scenarios. First, ML models definitely need enough high-quality training data. However, it is always prohibitively expensive to hire experts to acquire or label sufficient data, while relying on some other resources (e.g. web, crowdsourcing, rules) may result in noisy data and this data is likely to have a negative impact on ML tasks. Second, a large number of training data naturally makes training less efficient. Besides, many typical model structures are becoming larger and deeper, especially for deep learning models, which further exacerbates the problem of inefficiency. Third, model training is not a one-shot process, and data scientists always train iteratively, test many models and select the most appropriate one, but these trained models are hard to manage because they incorporate too much information (e.g. parameters, model structures, performance). Hence, how to manage these models and the corresponding data artifacts remains a challenge.

To address these challenges, an end-to-end data analysis pipeline is proposed, where many DB techniques can be utilized to benefit the effectiveness and efficiency of ML tasks. We discuss relevant techniques from three aspects along with the ML pipeline. First, before model training (Pre-ML), data preparation is needed for preparing high quality training data, including data discovery, cleaning and labeling. Secondly, model training (In-ML) consists of a number of time-consuming steps including feature selection, model selection as well as training and inference, which can be accelerated using DB techniques like materialization or parallelism. Thirdly, DB-based techniques can always be used to manage the trained models (Post-ML), including model storage, query, deployment and debugging. Thus the above steps have been extensively studied recently, and we review them thoroughly in this paper.

- *Chengliang Chai, Jiayi Wang, Yuyu Luo, and Zeping Niu are with the Department of Computer Science, Tsinghua University, Beijing, China. Email: {ccl@mail., jiayi-wa20@mails., luoyuyu@mail., niuzp20@mails.,}tsinghua.edu.cn*
- *Guoliang Li is with the Department of Computer Science, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing, China. Email: liguoliang@tsinghua.edu.cn.*
- *Corresponding authors: Yuyu Luo and Guoliang Li.*

1.1 Pre-ML: Data Preparation

Data preparation is the act of manipulating raw data from multiple data sources into a form that can be readily analyzed, e.g., feeding into an ML task. It is the first step in ML tasks and accounts for 80% time of the entire data science pipeline [2]. The quality of data preparation has a large impact on the model performance. In this part, we discuss three parts of modules along with the data preparation pipeline, i.e., data discovery, data cleaning and data labeling respectively.

Data Discovery aims to retrieve training data related to the ML tasks from external resources, like the web, data warehouse or data lake, which is the first step in data preparation. In this paper, we survey related works about data discovery from two aspects at a high level. (1) On the one hand, we review attribute/tuple-level data discovery [177], [169], which takes a dataset as input, asks for fresh attributes or tuples from other resources and fills them into the data set. The optimization goals are to achieve high accuracy (the fetched attributes/tuples belong to the domain of the dataset), and low cost (less human cost or rechargeable APIs). (2) On the other hand, we review table-level data discovery [75], [28]. Given a base table as input, it aims to discover multiple tables from data warehouse or data lake that can be joinable or unionable with the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

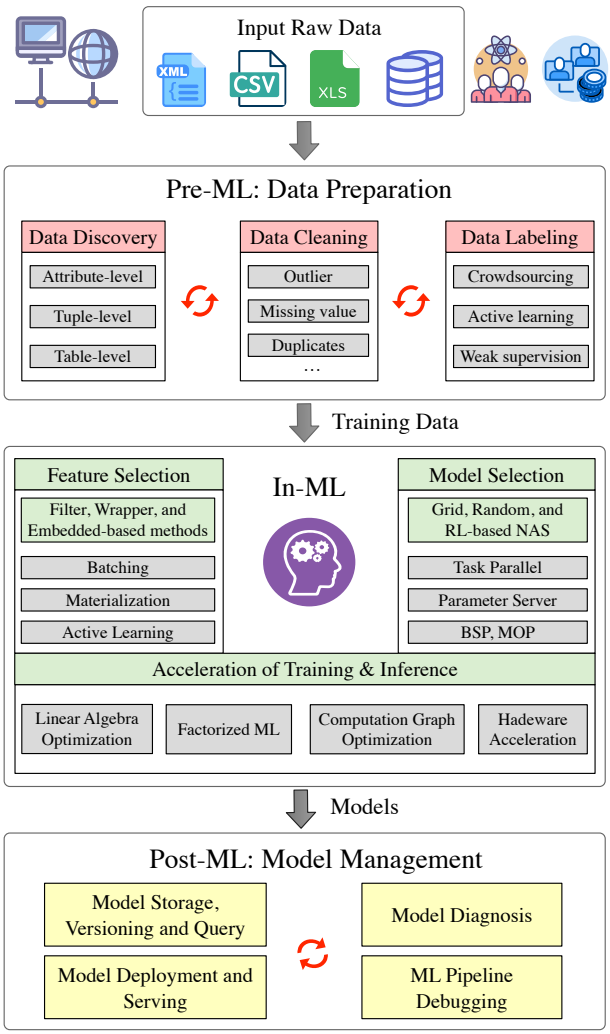

Fig. 1: Overview of Data Management for ML

base table. The optimization goals are mainly to improve the ML performance and the efficiency because of the time-consuming join operation. At a fine-grained level, we discuss whether these data discovery methods directly benefit the downstream ML task (ML-oriented) or not (Data-oriented), and focus on introducing ML-oriented approaches.

Data Cleaning [55] focuses on cleaning the dirty data that always happens in raw data collected from different resources. Common types of dirty data include missing values, outliers, duplicates, inconsistency, etc. In the literature of data cleaning, existing works always study how to detect and repair dirty data, with the goal of cleaning the data as much as possible. Recently, researchers have proposed to clean for ML, i.e., purely cleaning the data that can directly improve the model performance, which saves much cleaning costs while keeping high quality. In this paper, we survey different cleaning approaches (e.g., Active clean, Boost clean, etc.) for ML [70], [69], [64], [92], [16], with diverse dirty data types [97], [98] to be cleaned and different models to be benefited.

Data Labeling aims to label the data for training. As we know, a well-performed ML model needs high-quality labels, which is as least as important as an appropriate ML algorithm. In this paper, we discuss data labeling methods from the following three aspects. (1) Crowdsourcing [82] aims to leverage the intelligence of crowd workers to label the data items that are hard for the computer. Thanks to the crowdsourcing platforms like AMT [1], hundreds of thousands of workers can be hired for labeling the data. The optimization goals are workers' cost (because workers are not free), quality (workers are error-prone) and latency (workers are slower than the computer). (2) Active learning [78], [150] mainly relies on experts to label the data. The advantage is that experts always have high quality, but they are expensive. The challenging problem is how to select minimum number of data items to label so that a well-performed model can be derived. Many related approaches (uncertainty-based, query by committee) will be surveyed. (3) Weak supervision [134], [62], [132] builds connections between weak labels and downstream ML tasks. Based on where the weak labels come from, we classify weak supervision methods into three categories: a) Learn from the crowd, where the labels come from crowd workers. b) Data programming, the labels are derived from multiple rules designed by experts. c) Fact extraction, the labels are retrieved from external resources like knowledge.

1.2 In-ML: Model Training and Inference

After preparing the data, we aim to feed the data for training and then inference. In the ML community, a number of works [45], [57], [5] focus on accelerating this process, including feature selection, model selection and the computation in training/inference. In the DB community, there also exist some works focusing on this in a complimentary perspective or different scenarios. For example, some of them study how to implement ML algorithms inside the database such that the data is not necessary to be loaded outside the database, and thus the efficiency can be improved and the security can also be guaranteed. To be specific, we discuss the In-ML optimization in the DB community from following aspects.

Feature Selection is the first step of training, which selects an optimal subset of features for training. In the ML community, feature selection approaches are categorized into filtering-based, wrapper-based and embedded-based approaches, which will be briefly introduced in this paper. In the DB community, several techniques are proposed to accelerate the feature selection, especially the scenarios for relational data. (1) Batching [79] means to load multiple feature subsets to be tested into memory at once, so as to improve the I/O efficiency. (2) Materialization [174] achieves acceleration through storing some intermediate results in feature engineering while iterative training. (3) Feature pruning [75] eliminates the features that are likely not to benefit the ML models without training. (4) Active learning [6] based method aims to select a subset of high-utility data items to train.

Model Selection [116], [68], [84], [118] aims to select the most appropriate model (or model parameters) during training, which significantly influences the model performance. In ML community, model selection always conducts the neural architecture search to select optimal hyper-parameters or architectures. In DB community, parallelism techniques are utilized to accelerate model selection. The key idea of parallelism is to distribute the data/models to multiple workers and aggregate the results to compute the best model. Hence, the parallelism methods are complimentary to the ML-based ones, and they can be applied together to much accelerate the ML tasks.

Acceleration of model training & inference. In the ML community, existing methods accelerate the model training/inference from several aspects, such as hardware-based acceleration [114], data-efficient approaches [111], [112] and gradient computation optimization [66], [60]. In the DB community, we study this problem

from a different perspective where training data exists in multiple relational tables. The training and inference processes incorporate a number of complicated mathematical operations on one or more tables for computing the gradient and thereby minimizing the loss, so there exists large optimization potential for acceleration. We category this part as follows. (1) Linear algebra optimization [30], [53]. In ML, the training process always needs to optimize a loss function, where many linear algebra (LA) computations are involved. These computations are almost time-consuming matrix operations, so it is necessary to optimize them through a) In-database optimization converts the LA operations to be executed in databases so that DB-based acceleration methods can be utilized. b) Rule-based methods can be used for optimizing the LA, like rewrite or fusion of LA operators. c) LA operations can also be transformed to relation algebra, which has systematic optimization methods in DB community. (2) Factorized ML [74], [144]. In real scenarios, training data is stored on multiple relational tables and has to be joined when training, which leads to large computational costs. Therefore, factorized ML is proposed to push some computations to each single table for acceleration. (3) Computation graph optimization [59], [41] is a higher level optimization that focuses on optimizing the computational graph in deep learning through subgraph substitutions, so as to improve the runtime performance of a tensor graph. (4) Hardware acceleration [7], [152]. Modern hardware, like FPGA, GPU, can be utilized to accelerate the training and inference process by parallelism.

1.3 Post-ML: Model Management

While developing an effective and robust ML model, developers usually carry out dozens of model architectures, tuning their hyperparameters, and then checking the performance by training and testing. Once the model is developed, the ML practitioner will deploy the model and monitor its performance. If there are some unexpected results detected in the model monitoring phase, the developer may look back the find the root causes. Not surprisingly, manually keeping and tracking these models variants and associated data artifacts are hard, error-prone, and not scalable. Model management is responsible for storing, versioning, querying, deploying and debugging the ML models (including their metadata) effectively and efficiently.

Model Storage, Versioning and Query [161], [110] aim to store, log, search, and analyze the model variants and their metadata efficiently and effectively. Some works adapt column-store, compression, and indexing techniques to reduce the storage cost and improve query efficiency. There is a line of works that provide declarative query language or visual interface to make the users storing and querying their models and data easier.

Model Diagnosis [160], [51] is responsible for helping ML developers to understand why a training process does not achieve acceptable performance and assisting developers to find reasons in models/data. Since the model diagnosis usually touches the model parameters and data artifacts produced in the model developing process, a large volume of such data will cause two data management challenges: storage and computation. Thus, several techniques are proposed to reduce the storage cost such as deduplication and quantization. Some works adopt the sampling, materialization, and indexing approaches to reduce the computation cost (*i.e.,* reduce the execution time of model diagnosis queries).

Model Deployment and Serving [32] refers to deploy models in a production environment and serve models for prediction with low latency. Model deployment should provide developers with an easy-to-use infrastructure to deploy models in the production environment seamlessly without huge little effort. The key points of model serving are how to support low latency and high throughput model prediction, while guaranteeing the accuracy of prediction results.

ML Pipeline Debugging [173], [170] helps users to find the root cause of unexpected results after deploying models. Since the ML is the paradigm of "learning from data", the first step in ML pipeline debugging is data debugging. After double-checking the data, developers can move to model debugging (or code debugging). In this part, we survey the related works from the above two aspects.

1.4 Contributions

Comparisons with existing surveys. In this paper, we focus on the DB techniques that can directly improve the efficiency as well as effectiveness of ML models. Roh [138] studies data collection techniques for machine learning, mainly focusing on data preparation operations rather than the entire ML pipeline including acceleration and model management. Furthermore, our survey also emphasizes data management techniques that can directly improve the model performance. Kumar [72], [12] focuses on accelerating the model training step rather than improving the effectiveness. Besides, AutoML surveys [38], [52] review techniques in ML community mainly from the model perspective rather than the data.

To summarize, we make the following contributions (see Figure 1).

- We review Pre-ML techniques that utilize data preparation approaches to prepare high-quality training data with low cost, including data discovery, data cleaning and data labeling (see Section 2).
- We review In-ML techniques that utilize DB-based methods to accelerate each part of ML, including feature selection, model selection, acceleration of training&inference (see Section 3).
- We review Post-ML(model management) techniques including model storage & query, model diagnose, deployment and pipeline debugging in Section 4.
- We provide research challenges and future directions in Section 5.

2 Data Preparation

In this section, we survey how to prepare high-quality data before ML, including data discovery, data cleaning and data labeling.

2.1 Data Discovery

When a data scientist aims to build an ML model, she needs sufficient data for training and testing. If there is no data available, existing external data repositories(e.g. data lake, web) provide opportunities for data scientists to search the data. However, in real scenarios, usually only a small number of data is available, lacking attributes, tuples or features. To address this, many researchers have studied to discover more data for better model performance. In this subsection, we category data discovery as attribute/tuple-level and table-level discovery

Attribute/Tuple level. Infogether [177] leverages a vast corpus of HTML tables on the web to fill the missing attributes in the basic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

Category	Target	References	Sources	Model	Semantic Aware
Attribute/Tuple level	Data-oriented	[177], [186], [15]	Web	–	×
		[121], [39]	Human, KB	–	✓
		[169]	Hidden DB	–	×
	ML-oriented	[147]	Human	ALL	✓
		[33], [88], [117], [87]	Training Data	ALL	✓
Table-level	Data-oriented	[47], [46]	Data Lake	–	×
		[42], [43], [119], [187]	Data Lake	–	✓
	ML-oriented	[75]	Data Warehouse	NB, LR	×
		[151]	Data Warehouse	NB, LR, SVM, ANN	×
		[28]	Data Lake	ALL	✓

TABLE 1: Comparison of Data Discovery Methods

table T_b. The basic idea is to first identify web tables that match with T_b. Here the "match" means that they have the same type of entities and overlapping schemes. Second, for each missing attribute in T_b, aggregate the corresponding results from these matched tables and pick the most likely one. Wang et al. propose SmartCrawl [169], a novel method that discovers new attributes from external data sources. It fetches data from a hidden database (e.g., deep web) through a keyword-based API. Similarly, some other works [15], [121], [39], [177], [186] also study how to discover attributes from external resources.

Although the aforementioned works [177], [15], [121] can fetch tuples from external resources, their goal is just to enrich the table rather than directly improving the model performance. Active learning-based methods [147] find tuples that have a large impact on the current model, but the fetched tuples have to be labeled by humans, which will be discussed in detail in Section 2.3.2. For image data, augmentation operations, e.g., flip, rotate, scale, etc. are always applied to enrich the training dataset for performance improvement [33], [88], [117], [87]. For example, AutoAugment [33] uses reinforcement learning to search a sequence of optimal augmentation policies automatically. Specifically, each policy defines which operations to use and the probability of applying the operations in each batch. The agent first predicts a policy from the pre-defined search space. Then a child network is trained to achieve certain accuracy after applying the policy. Finally, the accuracy is regarded as a reward to update the prediction strategy in the first step.

Table level. Goods [46], [47] is a table-level method, which manages the data in the data lake and provides an interface that allows users to explore the tables. Specifically, it stores information like dataset scheme, similarity and provenance between datasets. Users can search and browse datasets using keywords. However, such systems have limited usage in ML because they can not support to search related datasets given a query dataset, which is a common requirement when current data is not enough. To this end, Fernandez et al. propose Aurum [42], which is a graph-based data discovery system that provides flexible queries to search datasets based on users' requests, so that one can find candidate datasets to join. The system consists of three parts. (1) *Graph Building.* Aurum leverages enterprise knowledge graph (EKG) to capture a variety of relationships between datasets. The EKG is a hypergraph where each node denotes a table column, each edge represents the relationship between two nodes and hyperedges connect nodes that are hierarchically related such as columns in the same table. Based on EKG, they have built the relationships between datasets in the data lake, and thus one can query related datasets using a dataset. (2) *Graph Maintaining.* Since data is always changing, keeping the graph up-to-date is significant, but each time recomputing the graph from scratch is expensive. Therefore, Aurum proposes a sampling-based method to determine which dataset has been changed efficiently, and then update that parts of the graph. (3) *Graph Query.* Given EKG, Aurum allows users to query it with property or relation constraints, such as querying tables using keywords, querying similar tables or ranking the results. The core part of EKG is whether the edges capture the internal relationships between relational tables accurately, so Fernandez et al. [43] propose to use word embeddings to capture the semantic relationships between relational tables. Specifically, they encode the tables name, column schemes, and map them onto an existing ontology. Two tables corresponding to two nodes of an edge in an ontology means that they have a close relationship.

Nargesian et al. [119] focus on searching tables from the data lake that can be unionable with the base table. It first defines the attribute unionability that measures whether two attributes can be in the same domain, which considers (1) *Set domains*: the co-occurrence of the same value in two attributes. (2) *Semantic domains*: even if values in two attributes do not overlap, they can map to an ontology (e.g. attributes including cities, basketball players). (3) *Natural language domains* : since existing ontologies cannot cover all domains, word embeddings are incorporated to capture the attribute unionability. Then given two tables S and T, if there is a one-to-one alignment between subsets attributes of S and T such that the aligned attributes are unionble, S and T are unionble.

The above works focus on finding related tables that can be joinable or unionable with the base table, rather than improving the downstream model performance directly. Next, we discuss table-level data discovery for ML directly. Formally, given a base table T_b (with a column Y for prediction), a set of candidate tables $\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$ that can be joined with T_b and a model M. M can be applied on T_b or augmented T_b by joining with tables in $\mathcal{T}$ and test the accuracy. The problem is to select an optimal subset of tables $\mathcal{T}' \subset \mathcal{T}$ and join all tables in $\mathcal{T}'$ with T_b, so as to improve the model performance most.

Halmet [75] answers a question that whether key-foreign key joins (KFK) between T_b and tables in $\mathcal{T}$ are necessary with the goal of improving the performance of M. If not, some tables in $\mathcal{T}$ can be pruned in advance without joining and training, so that the efficiency is improved. The basic idea is that the foreign key in the base table has been encoded enough information of candidate tables. For example, suppose that the base table T_b is `Customers (CustomerID, Favor_type, Age, Gender, MovieID)` , and one of the candidates is `Movies (MovieID, Rating, Type)`, so they can be joined through `MovieID`. Since many customers might see the same movie, so it is reasonable to use `MovieID` as a feature directly rather than joining the `Movies` table. Specifically, Halmet proposes a measurement called Risk Of Representation (ROR) that leverages the

Vapnik-Chervonenkis (VC) dimension to compute the difference of train and test error of M if the join is removed. The higher the ROR is, the more necessary the join will be. Otherwise, one may consider to drop the candidate table. However, this method is limited to models like Naive Bayes and logistic regression, with VC dimension linear to the number of features. Therefore, Shah [151] proposes a framework that can avoid unnecessary joins based on high-capacity classifiers such as decision trees and SVMs.

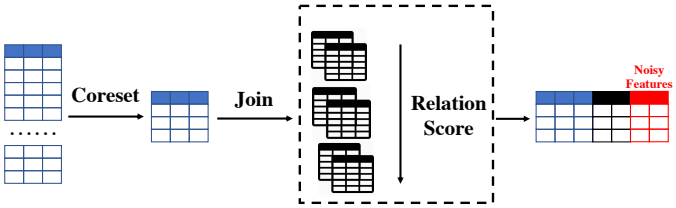

Fig. 2: ARDA Framework

ARDA [28] is an ML-oriented data augmentation framework that discovers tables from data lake, so as to improve the performance of the downstream ML model. More specifically, given a base table T_b and a large number of tables that can be joined with T_b, the most straightforward method is to join all tables and then apply feature selection algorithms on the join results. However, this method is too expensive because the join operation is costly. What's more, adding many features is likely to introduce noise, leading to the decline of model performance. Therefore, ARDA proposes the following methods to address this, as shown in Fig. 2. *(1) Base table sampling:* since T_b is required to be joined multiple times, reducing its cardinality can be a way to improve the efficiency. Therefore, ARDA samples some representative tuples as a coreset using simple heuristics from T_b when its size is large and utilizes these samples to learn the augmentation plan. *(2) Step-by-step fuzzy join:* ARDA further improves the efficiency by joining T_b with candidate tables step by step rather than joining all of them together. First, it sorts the candidates based on their relevance to T_b, which can be computed by [42]. Then it joins the table step by step based on the ordering. In each step, ADRA feeds a batch of tables to join with T_b considering the trade-off between storage overhead and efficiency, conducts feature selection, trains the model and tests the accuracy. When the accuracy does not increase or a pre-defined budget of steps has been achieved, the augmentation process terminates. What's more, ARDA can handle fuzzy joins by soft keys. *(3) Random injection feature selection (RIFS):* to verify the effectiveness of newly added features, ARDA compares these features against noise. If the features perform worse than injected noise, they are likely to be useless.

2.2 Data Cleaning

In the real world, most of the data is dirty, which may result in unreliable analysis and decisions made by the ML model. Common types of dirty data include outliers, missing values, inconsistency, etc. Therefore, it is necessary to clean the data. The pipeline of cleaning the data consists of error detection and error repair (see [55] for a survey). Some researchers use rule-based methods to detect and repair [40] data errors, which are easy to implement, but have limited accuracy and generality. Therefore, some works have studied to leverage humans (e.g., crowdsourcing or experts) and external resources (e.g., knowledge bases) [29] to improve the cleaning quality. However, humans are always expensive, so some statistical techniques [135] are applied to clean the data. Most data cleaning approaches mentioned above mainly focus on cleaning the entire dataset. However, data cleaning is task-dependent and different tasks may use different cleaning techniques to repair different parts of the data. Next, we introduce some works that focus on cleaning the data for downstream ML or data analysis tasks.

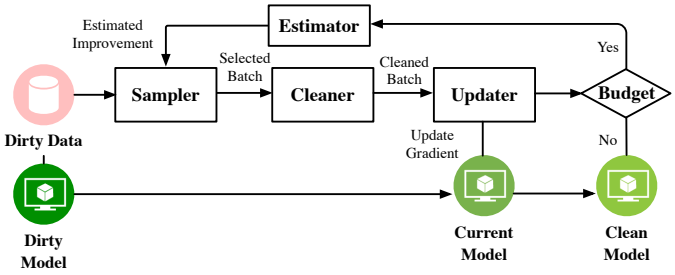

Fig. 3: The Framework of ActiveClean

Given a dataset with dirty data and a model, Krishnan et al. [70] observe that iteratively cleaning the dataset and training on a partially cleaned dataset is likely to degrade the model performance. One can also choose to clean the entire dataset using the above methods, but it is always human expensive. To this end, Krishnan et al. [70] propose ActiveClean to prioritize cleaning those dirty records that have a large impact on the downstream model in iterations. At a high level, the reason why dirty data degrades the performance is that they mislead the gradient computation. Then, if one can clean part of dirty data, compute the variations of gradients and estimate the gradients of uncleaned data, quick convergence with a smaller number of cleaned data can be achieved. To be specific, as shown in Fig. 3, ActiveClean consists of four modules, which are Sampler, Cleaner, Updater and Estimator respectively. Sampler is utilized to select a batch of records to be cleaned, where the selection criterion is measured by how much improvement can be made after cleaning a record, i.e., the variation of the gradient. The improvement is computed by an Estimator based on current cleaned data. Then the selected records will be checked and repaired by the Cleaner, which can be conducted by humans. Next, the Updater updates the gradients of the entire dataset based on these verified dirty data. The above four steps are repeated until the budget is used up. Note that ActiveClean only supports cleaning outliers and string normalization. Besides, ActiveClean focused on models with convex loss (e.g., Logistic regression, SVM) because global optimum can be guaranteed in this situation.

Krishnan et al. propose BoostClean [69] to clean the data where an attribute value is out of range. It takes as input a dataset, a set of pre-defined functions for detecting and repairing out-of-range values, as well as a black-box model. These values will definitely degrade the model performance, so the optimization goal of BoostClean is to explore the best sequence of repair operations so as to best improve the model, rather than costly enumerating every record in the original dataset and clean it. It makes the observation that detecting errors and applying a cleaning function on a dataset can be seen as generating a set of new features, and then a new model can be trained on the cleaned dataset. Therefore, the basic idea of BoostClean is to regard the best repair sequence selection problem as an ensembling problem, which applies multiple repair operations on each detected dirty record

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

Method	Target	Method	Automatic	Cleanning Type	Model Type
ActiveClean [70]	Model ↑	Gradient estimation	×	Outlier, Normalization	Loss convex model
BoostClean [69]	Model ↑	Boosting	✓	Outlier	All
CPclean [64]	Model ↑	Incompletion analysis	✓	Missing values	KNN
DAGAN [92]	Model ↑	GAN	✓	Missing values	ALL
SampleClean [165]	AQP ↑	Statistics	×	Outlier, Duplicates	Aggregation queries

TABLE 2: Comparison of Data Cleaning Methods

and infer the final result. To be specific, it consists of two modules, i.e., error detector and repair selector. The former one takes as input some pre-defined error detection rules, applies these rules on records and generates a numerical vector for each record. Then a sophisticated outlier detection technique(Isolation Forest [90]) is used to detect errors (outliers) based on these feature vectors. Then the latter one will select cleaning operations to apply on these detected records. Inspired by a typical method, Boosting [143] in ensemble learning, BoostClean views each repair operation as a weak classifier, and combines many of them collectively to make predictions. To be specific, in each training iteration, it first generates the best classifier on the current dataset, then weights the dataset considering the mispredictions by current classifiers and repeats the above steps until the budget (pre-defined maximum number of cleaning operations) is used up.

CPClean [64] focuses on reasoning about the impact of missing values on the downstream ML task M. In another word, given a test example t, CPClean studies how much impact the missing values have on the prediction $M(t)$. To be specific, suppose a dataset D has n missing values and the domain size is m, we have m^n possible imputations. All the imputations constitute a possible world and each of them corresponds to a model. We hope that the predicted labels of all models are relatively consistent for each t so that fewer cleaning operations are necessary. An ideal case is that all labels are the same for an example t, then t is called, *certain prediction (CP)*. Intuitively, if all test samples are CP, it is wasteful to impute the missing values. Therefore, the optimization goal of CPClean is to prioritize imputing the records that are likely to make test samples have consistent labels, towards to be CPs. To this end, one has to obtain the labels of models in the possible world. A straightforward method is to train all the models and make predictions, which is too expensive because training is inefficient and the possible world is extremely large. CPClean addresses this problem by restricting the model to *KNN classifier*. In this way, to compute whether t is CP, we only need to consider the similarity scores between t and training data, rather than training models.

DAGAN [92] focuses on the difference of the distribution of missing values on multiple attributes, between training data and test data. This often causes performance degradation on test data and it is a common scenario because a single trained model is likely to be used in different test cases with different missing value distributions. To address this, the basic idea of DAGAN is to reduce the divergence on data distribution between training data and test data. Firstly, it captures the missing value distribution from test data, then it adapts the training data to the captured distribution without changing the labels. After that, it refines the ML model by retraining with adapted data. In order to make the adaptive learning process more efficient, two connected GANs are adopted. One learns the distribution from test data, and the other learns the observed data distribution to augment the training data. Compared with CPClean, DAGAN is not restricted to a certain model.

Method	Positive	Negative	Model Selection
Duplicates	-	✓	✓
Outlier	✓	✓	✓
Inconsistency	-	-	✓
Mislabels	✓	-	-
Missing values	✓	-	✓

TABLE 3: CleanML Summarization

SampleClean [64] is proposed to clean the data so as to obtain a high-quality approximate query processing (AQP) result for aggregation queries on databases. Although it is not for ML directly, aggregation queries are important to data analysis [100], [101], [128] and can also be used as a feature for downstream ML tasks. This paper mainly considers errors including numerical outliers and duplicates. For example, an outlier will have a large impact on aggregation operations (*e.g.,* Average, MIN/MAX and SUM). However, cleaning the entire dataset and then applying queries on the cleaned data are expensive, so SampleClean uses an sample-and-clean framework to solve the problem. The basic idea is that it first samples a subset of data randomly, and then uses data cleaning techniques to clean the data. After that, SampleClean leverages the cleaned data to answer aggregation queries using statistic techniques. For example, consider we sample a subset S from D and there is a SUM query with a predicate θ. Given a tuple t, $\theta(t) = 1(0)$ denotes t satisfies (dissatisfies) the predicate. Then we clean on S and estimate the result over D as $\frac{1}{|S|}\sum_{t\in S}\theta(t)\cdot|D|$. They also prove that the estimated results are unbiased.

CleanML [85] is a benchmark for joint data cleaning and ML. It collects 13 real-world datasets containing 5 different types of errors, which are outliers, duplicates, inconsistencies, mislabels and missing values. CleanML tests the impacts of these errors on 7 typical machine learning models (*e.g.,* logistic regression, decision tree, random forest, etc). To summarize, we draw a table to illustrate the relationships between errors and ML performance.

In Table 3, the first column lists the error types. The second (third) column shows whether data cleaning has a positive (negative) impact on machine learning tasks. The last column means that whether model selection can further improve the performance when data cleaning has positive impacts, or eliminate the negative impacts. For example, the first row indicates that cleaning duplicates will degrade the model performance, but the degradation can be alleviated by judiciously selecting models. Cleaning the outliers may have positive or negative impacts on ML tasks, which highly depends on the datasets. Making the data more consistent is not likely to influence the model, but model selection increases the probability of having positive impacts. Verifying mislabels and imputing missing values have positive impacts on ML tasks.

2.3 Data Labeling

ML models always need data with labels for training, especially for supervised machine learning. As we know, the quality of labels has a large impact on the model performance. However, high

quality training data is always hard to derive because it is always human expensive. In this section, considering the labeling quality, human costs, we review several directions for data labeling, including crowdsourcing, active learning and weak supervision.

2.3.1 Crowdsourcing

Crowdsourcing [18] leverages the crowd intelligence to solve the tasks that are hard for the computer (See survey [82] for details). Thanks to the crowdsourcing platforms like AMT [1], hundreds of thousands of crowd workers can be hired to process users' tasks, like image classification, entity resolution or semantic tagging. There are three factors in crowdsourcing that should be optimized. *(1) Quality* [104], [172], [185], [20], [21]: Crowd answers may not be reliable because malicious workers randomly return answers or some tasks are difficult for humans to answer. *(2) Cost* [35], [164], [166], [23], [22], [17], [49]: Humans are not free, but the budget of a user is always limited. Hence, some techniques have to be utilized to save the monetary cost. *(3) Latency* [142], [162]: Since workers need time to think and answer, they will be much slower than the machine, so it is necessary to reduce the latency. There exist trade-offs among the above three factors [80], [19], [81]. For example, asking more workers to answer a task may increase the quality, which costs much.

2.3.2 Active Learning

Active learning is a commonly used technique in machine learning, which involves experts to label the most interesting examples iteratively [127], [126]. Unlike crowdsourcing, it always assumes that humans can provide accurate answers. The key challenge is that given a limited budget, how to select the most appropriate examples in each iteration. Active learning has been covered extensively in surveys [147], [4], so we only cover the most prominent techniques in this part. Next, we will introduce several strategies of selecting items to be labeled in each iteration.

Uncertainty sampling. Uncertainty sampling [78] is one of the simplest and most commonly used methods in active learning, which selects the next unlabeled example that the current model regards as the most uncertain one. For example, when using a probabilistic model for binary classification, uncertainty sampling chooses the example whose probability is the nearest to 0.5. The uncertainty is always measured by the entropy.

Query-by-committee (QBC). The QBC [150] approach extends uncertainty sampling by maintaining a committee of models which are trained on the same labeled data. Each committee member can vote when testing each example, and the most informative example is considered to be the one that most models disagree with each other. The fundamental idea is to minimize the version space, which is the space of all possible classifiers that give the same classification results as the labeled data.

Decision-theoretic approach. Another active learning framework utilizes the decision-theoretic approach, choosing the example that would introduce the greatest change to the current model with the assumption that the label is known. It mainly consists of two approaches. The first one is the expected gradient length (EGL) approach [149], the change to the model can be measured as the length of training gradient. In other words, we should select the example that will lead to the largest gradient if it is labeled. The latter one [139] aims to measure how much its generalization error is likely to be reduced rather than how much the model is likely to change. Given an example, the basic idea is to first estimate the expected future error of the model trained using the example together with current labeled data on the remaining unlabeled examples. Then the example induced the smallest error is selected. However, for the above two methods, the true label is not known in advance, so an expectation should be optimized.

Density-weighted methods. The above frameworks are likely to choose the outlier examples, which might be uncertain and disagreeing but not representative. However, most time the outliers contribute less than the representative examples which follow the similar distribution of the entire dataset. Therefore, existing works [148], [176] focus on choosing examples not only uncertain or disagreeing, but also representative of the example distribution.

2.3.3 Weak Supervision

In the above sections, crowdsourcing-based algorithms focus on improving the quality of the training data itself, while neglecting the impact on the downstream model. Active learning methods can choose items to label considering the model performance, but it is prohibitively expensive to hire experts. To address these limitations, weak supervision approaches are proposed to build connections between weak labels and supervised ML models, where these labels can be generated from crowdsourcing, human-crafted rules or knowledge bases. Next, based on where weak labels come from, we talk about learning from imperfect labels, data programming and fact extraction.

Learning from imperfect labels. In this part, connections are built between imperfect labels and supervised models. Raykar et al. [134] propose a general framework for this problem. Specifically, given a ML task, a typical setting is that we have a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where each x_i is a tuple and y_i is the true label of x_i. However, in real scenarios, these true labels are not known in advance and needed to be labeled, denoted by $\{\hat{y_i}\}_{i=1}^N$. Hence, taking $\{(x_i, \hat{y_i})\}_{i=1}^N$ as the observations, Raykar [134] used maximum likelihood estimation to infer the model parameters $\theta = \{w, \alpha, \beta\}$ based on the answers of h labelers, which maximizes:

$$P(\mathcal{D}|\theta) = \prod_{i=1}^{N} P(\hat{y_1^1}, ..., \hat{y_i^h}|x_i, \theta),$$

Then the above equation is transformed into the following one, considering the true labels $\{y_i\}_{i=1}^N$ as hidden variables, which can be solved by the Expectation Maximization (EM) [115] algorithm, where $\alpha(\beta)$ is the true(false) positive rate.

$$\begin{aligned} P(\mathcal{D}|\theta) = \prod_{i=1}^{N} & P(\hat{y_i^1}, ..., \hat{y_i^h}|y_i = 1, \alpha)P(y_i = 1|x_i, w) \\ & + P(\hat{y_i^1}, ..., \hat{y_i^h}|y_i = 0, \beta)P(y_i = 0|x_i, w). \end{aligned}$$

Besides, many other methods have been proposed to build models from imperfect labels directly without truth inference. Kajino et al. [62], [61] propose to learn a logistic regression model directly from crowd workers. Specifically, each worker is treated as an independent classifier, and all classifiers are modeled as a multi-task learning paradigm that can be solved by convex optimization. Zhou et al. [189] define the confusion matrix for each item-worker pair, which can capture the difficulty of each item and provide more fine-grained optimization. Platanios et al. [124] used a generative model to build relationships between workers and items, and [123] extended it to support categorical labels rather than purely binary.

Method		References	Label Quality	Label Cost	Label Latency	Model-aware
Crowdsourcing		[104], [172], [35], [164], [142]	Medium	Medium	Medium	×
Active Learning		[78], [150], [149], [139], [148]	High	High	High	✓
Weak Supervision	Learn from imperfect labels	[134], [62], [61], [137], [8]	Medium	Medium	Medium	✓
	Data programming	[131], [132], [141], [93], [10]	High	Low	Low	✓
	Fact extraction	[140], [113]	Medium	Low	Low	×

TABLE 4: Comparison of Data Labeling Methods

In recent years, with the rapid development of deep learning, there exist some works that apply deep learning to learn from noisy labels. For example, Rodrigues et al. [137] also use EM algorithm to jointly learn the parameters of the network and the quality of crowd workers, but they propose a crowd layer that can train neural networks end-to-end directly from the noisy labels provided by multiple workers using backpropagation. Atarashi [8] learns from both imperfect labels and unlabeled data using deep semi-supervised learning. To be specific, latent features and data distribution of unlabeled data are considered and a generative model is applied to train a classification model.

Data programming. Many ML applications need a large number of training data, which is prohibitively expensive to obtain, even resorting to crowd workers. Therefore, data programming techniques have been proposed to generate a large number of weak labels using multiple labeling functions, where each function can be written by humans. Naturally, a single function is far from generating high-quality labels, so multiple functions should be combined to generate labels. The most straightforward method of combination is majority voting, but it does not consider the correlations and qualities of different functions.

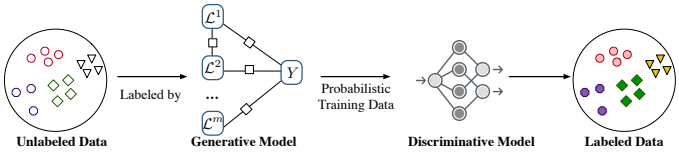

Fig. 4: Snorkel Framework

To address this, Snorkel [131], [130] is proposed to aggregate the results of multiple labeling functions to infer the final labels. To be specific, as shown in Fig. 4, Snorkel first provides the user-friendly interface to incorporate different types of labeling functions, and applies these functions on data gives us a label matrix $\mathcal{L}$, where $\mathcal{L}_i^j$ denotes the label assigned by the j-th function on the i-th item. Second, following the techniques in [9], [133], Snorkel leverages the factor graph to build a generative model $p_w(\mathcal{L}, Y)$, where $Y = \{y_1, ...y_i, ..y_m\}$ denotes the unknown true labels. The factor graph aims to model the labels in $\mathcal{L}$ for accurate inference. Concretely, three factors, including the labeling cover, accuracy, and pairwise correlations of labeling functions are considered, denoted by $\phi_{i,j}^{cov}(\mathcal{L}, Y) = \mathbb{I}\{\mathcal{L}_i^j \neq \emptyset\}$, $\phi_{i,j}^{acc}(\mathcal{L}, Y) = \mathbb{I}\{\mathcal{L}_i^j = y_i\}$ and $\phi_{i,j,k}^{cor}(\mathcal{L}, Y) = \mathbb{I}\{\mathcal{L}_i^j = \mathcal{L}_i^k\}$, where $\mathbb{I}\{\cdot\}$ is an indicator function that $\mathbb{I}\{\cdot\} = 1$ if the condition inside is satisfied, and otherwise $\mathbb{I}\{\cdot\} = 0$. $\phi_{i,j}^{cov}$ denotes that the j-th function does not abstain the i-th item. $\phi_{i,j}^{acc}$ denotes that j-th function predicts accurately, but y_i is taken as a latent variable because it is unknown. $\phi_{i,j,k}^{cor}(\mathcal{L}, Y)$ represents that j-th and k-th functions are highly correlated. Overall, the model is defined as:

$$p_w(\mathcal{L}, Y) = Z_w^{-1} \exp(\sum_{i=1}^{m} w^T \phi_i(\mathcal{L}, y_i)),$$

where Z_w is the normalizing constant, ϕ_i is the concatenated vector of the above three factors for the i-th item and w is the corresponding parameters to be optimized. Since y_i is unknown in advance, we obtain w by optimizing the marginal likelihood over Y. Afterwards, the probabilistic training labels $\hat{Y} = p_w(Y|\mathcal{L})$ can be inferred through the factor graph model. Then finally, Snorkel trains a discriminative model over the input items and these probabilistic training labels to generalize beyond the labeling functions. Following the Snorkel project, Ratner et al. [132] have extended Snorkel for multi-task learning. Mallory et al. [106] use Snorkel to extract chemical reactions from text. Bach et al. [131] study how to deploy the Snorkel framework for an industrial scenario in Google.

Moreover, some existing works [178], [93], [10] utilize humans to qualify the labeling rules and leverage the high-quality rules to label the data. Fan et al. [178], [93] propose CrowdGame, which generates good rules to reduce the labeling cost while achieving high quality. It first generates a candidate set of rules. Then a group of crowd workers verify these rules, served as rule generators. Another group of workers check the tuples covered by the rules, served as rule refuters. The two groups of workers work together iteratively to generate high-quality labeling rules. Boecking et al. [10] propose an interactive weak supervision(IWS) framework. Similar to CrowdGame, IWS generates a family of rules and asks experts to verify them. Then a set of high-quality rules are selected to build an end-to-end classifier like Snorkel.

Fact extraction. Fact extraction is another way to generate weak labels using knowledge base, which contains facts extracted from different sources. A fact usually describes entities with attributes and relations, such as `<China, capital, Beijing>`, which indicates the capital of China is Beijing. The facts can be regarded as labeled examples, which can be used as seed labels for distant supervision [140]. The Never-Ending Language Learner (NELL) system [113] continuously extracts structured information from the unstructured Web and constructs a knowledge base. Initially, NELL starts with seeds that have an ontology of entities and relationships among them. Then NELL explores large quantities of Web pages and identifies new entity pairs, which have the same relationships with seeds based on the matching patterns. The resulting entity pairs can then be used as the new training data for constructing more patterns.

3 Model Training & Inference

In this section, given sufficient and high-quality training data, we focus on how to train a well-performed model while keeping high efficiency on both training and inference stages. To be specific, we will discuss feature selection (Section 3.1), model selection (Section 3.2), acceleration approaches of training and inference using database techniques (Section 3.3).

3.1 Feature Selection

Given a set of features $\mathcal{F} = \{f_1, f_2, ..., f_m\}$ on a dataset, feature selection(FS) aims to select an optimal subset $\mathcal{F}^* \subset \mathcal{F}$ so that a model trained on $\mathcal{F}^*$ achieves the best performance. Usually,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

Goal	Methods	Quality	Efficiency	Model
For Model Quality	Filtering [45]	Low	High	All
	Wrapper [52]	High	Low	All
	Embedded [57], [156], [77]	High	-	Lasso, CART, DNN, etc.
For Model Effciency	Batching [79]	-	Medium	All
	Materialization [79], [174]	-	Medium	All
	Pruning [75], [151]	-	High	NB, LR, SVM, ANN
	Active Learning [6]	-	Medium	All

TABLE 5: Comparison of Feature Selection Methods.

feature selection approaches can be categorized into *filtering methods*, *wrapper methods* and *embedded methods*.

Filtering methods. This class of methods ranks features in $\mathcal{F}$ independently using some score functions and select the top-ranked ones as $\mathcal{F}^*$. Typical functions [45] include mutual information, correlation coefficient between the features, variance, student's t-test or Fisher test. The advantage of this method is its efficiency because it does not need iterative training. The disadvantage is the selected features may not be optimal because the dependencies of features are not considered and it does not account for the variation of model performance for feature subsets.

Wrapper methods. To address the disadvantage of filtering methods, wrapper methods are proposed to use the predictive model to score feature subsets. In general, wrapper methods consist of the following steps. (1) Generate a feature subset $\mathcal{F}'$ and the corresponding dataset $D_{\mathcal{F}'}$. (2) Evaluate $\mathcal{F}'$ by building an ML model on $D_{\mathcal{F}'}$. (3) Iteratively repeat the above two steps until a predefined performance is achieved or all subsets have been evaluated. As wrapper methods train a new model for each subset, they are computationally intensive ($O(2^n)$), but usually get the best performing feature set for a particular type of model or typical problem and they are very commonly used in model training.

To address the efficiency problem, in the ML community, many approaches have been proposed to reduce the search space by generating some candidate feature subsets (see [52] for a survey). Recently, several works [79], [174], [75], [151], [6] have been proposed to leverage the DB optimization techniques to accelerate the FS process from feature subsets enumeration and feature subsets evaluation, which are the focus of this subsection.

Embedded methods. This class of methods integrates the feature selection into the entire training process. The exemplar is LASSO [57] for building a linear model, which penalizes the regression coefficients with an L1 penalty, shrinking many of them to zero. Besides, Decision trees such as CART [156] have a built-in feature selection mechanism. Recently, deep learning methods [77] have shown powerful capability on automatic feature selection, but they need a large training dataset.

Next, we introduce how to accelerate the feature selection using database techniques, including batching, materialization, pruning and active learning.

Batching. Traditional feature selection tasks often need to scan the same table many times, which is very time-consuming. For example, given two subsets $\mathcal{F}_1 = \{f_1, f_2, f_3\}, \mathcal{F}_2 = \{f_2, f_3\}$ to be evaluated. Without any optimization, D_F has to be scanned twice. The batching technique used in Columbus [184] is to batch some feature subsets together and evaluate them within a single table scan, so as to reduce the I/O costs. Each subset $\mathcal{F}_i$, corresponding to an ML model, has a memory requirement m_i for training. Therefore, given a memory size constraint S, the sum of memory usage of subsets evaluated in each batch should not exceed S. Given the constraint, the batching optimization problem can be modeled as the bin-packing problem, which has been proved to be NP-Hard. Therefore, Columbus adopts a heuristic method [79] to solve the problem. The main idea is to select a subset as a batch randomly, add other subsets one by one into the batch until the memory constraint is satisfied and then construct a new batch until all subsets are put into a batch.

Materialization. Given batches of feature subsets, a straightforward method is to load each batch via a table scan and then evaluate them. However, as some small number of features are accessed frequently, Columbus [184] further reduces the costs through materializing some tables. For ease of representation, we use the notation B to denote the union of subsets of features in a batch. Suppose $B_1 = \{f_1, f_2, f_3\}$ will be evaluated by 10 epochs; while $B_2 = \{f_1, f_2, f_4\}$ will be evaluated by 100 epochs, which is visited more frequently than B_1. Given this information, we may materialize the columns corresponding to the features in B_2. Then when we evaluate B_2, we do not need to load the entire table. Specifically, given a set of batches $\mathcal{B} = \{B_1, B_2, \cdots, B_N\}$ to be evaluated, Columbus aims to select an optimal materialized solution $\mathcal{M} = \{M_1, M_2, ...M_K\}$ to minimize the evaluation cost, where each $M \in \mathcal{M}$ denotes a feature subset, whose corresponding columns will be materialized as a table. The cost can be derived from the sum of following aspects. (1) the materialization cost of $\mathcal{M}$, including reading tables from the disk and write the materialized parts back; (2) for each batch $B \in \mathcal{B}$, i) if it can be read from $M \in \mathcal{M}$, the cost equals to the read cost of M. ii) the cost is the read cost of the entire table concerning to the features in B. The problem is proved to be NP-hard reduced from the set cover problem [79]. Next, Columbus focuses on a frequent case in the FS process, where batches in $\mathcal{B}$ can constitute a chain, i.e., $B_1 \subseteq B_2 \subseteq \dots B_N$. In this case, Columbus proposes a dynamic programming algorithm to solve the optimization problem with a time complexity of $O(N^2)$.

Given a dataset, the above method assumes that features have already been extracted and processed, the only thing that needs to be done is to select an optimal subset of features. However, in real applications, given the raw data, many steps with respect to feature processing are necessary such as feature extraction, feature transformation, feature concatenation. From this perspective, Xin et al. [175], [174] propose Helix that optimizes the entire ML workflow, with much focus on feature engineering, through materialization. To be specific, Helix models the ML flow as a directed acyclic graph (DAG), where each node denotes an operation, like feature extraction on an attribute. A directed edge denotes that an operation is applied after another. At a high level, multiple series of operations on different features comprise multiple paths in the DAG. The intersection of paths represents that different features are combined together for a more complex representation. Hence, each ML iteration is to implement the nodes in topological order. Naturally, an ML task needs multiple iterations, each of which is likely to induce some modifications to the workflow. Therefore, it is prohibitively expensive to implement the DAG from scratch in each iteration. To address this, Helix proposes to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

materialize the results of some nodes, with the goal of accelerating feature engineering during the entire ML iterations. Specifically, considering the time of recomputing a node (including the time of recomputing its ancestors, if not materialized), the time of loading the node (if materialized) and the storage overhead, Helix models the materialization problem as an NP-Hard problem and uses a heuristic algorithm to judiciously select features to materialize.

Feature Pruning. Existing works [75], [151] discussed in Section 2.1 can also be utilized to select features. They mainly focus on the situation where features are distributed on multiple tables, and thus the join operation should be conducted for feature selection. They leverage the VC dimension in machine learning theory to efficiently prune features that are not effective for the ML model without training.

Active learning. For each feature subset, traditional methods will utilize all items corresponding to these features to train an ML model and evaluate the performance. To accelerate the feature subset evaluation process, Zombie [6] proposes to only use part of items with high *utility* on model training, leveraging the idea of active learning. These items with high *utility* can accelerate the model convergence. The utility of an item is measured by upper-confidence bounds (UCB), which is the upper bound of a certain confidence interval. The higher the UCB of an item is, the more uncertain and informative the item is. More specifically, firstly, the input data will be clustered into several groups based on their distribution and thus data in each group will have high similarity. Secondly, ZOMBIE computes the average utility of items in each group as the utility of the group, greedily selects the group with the highest utility and randomly samples an item from it to train an updated model. Thirdly, given the new model, ZOMBIE updates the utility of each item and repeats the above steps until converge.

3.2 Model Selection

Model selection aims to generate a machine learning model and set the hyper-parameters to improve the model performance most for a specific ML task [73]. Existing methods consist of two directions, traditional model selection and neural architecture search (NAS). The former one focuses on selecting the best model from traditional approaches such as SVM, Random Forest, KNN, etc, while NAS aims to automatically construct a well-performed neural architecture, including model structure design and hyper-parameter settings, which is a current hot topic in both ML and DB community. In the ML community, many automated techniques (see [5] for a survey) like grid/random search, reinforcement learning method, Bayesian optimization have been proposed to achieve well-performed models or improve the efficiency of training one model at a time. However, the key bottleneck of the model selection problem is its throughput, i.e., the number of training configurations tested per unit time. High throughput allows the user to test more configurations during a fixed period, which makes the entire training process more efficient. Therefore, in this subsection, we focus on the DB-based acceleration for NAS, where parallelism is the main technique.

Task Parallel. Moritz et al. [116] propose a distributed framework Ray, which allows different training configurations to run on different workers in a task parallel way. As shown in Fig. 5(a), in each worker node, sequential SGD is applied on the entire dataset and thus the reproducibility and convergence can be guaranteed. However, the downside of this method is that it has to copy the whole dataset to each worker, which results in poor scalability.

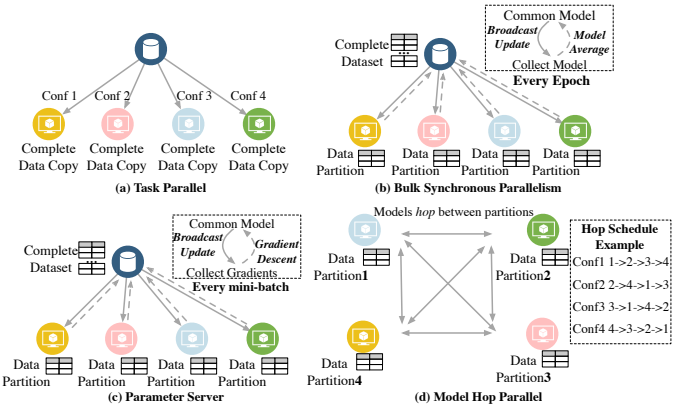

Fig. 5: Parallel Model Selection Methods

If there is a large dataset, a user has to sample the data to train, which may result in overfitting. Moreover, transferring the whole dataset to workers also leads to waste of memory and storage. To address the scalability issue, bulk synchronous parallel approach is proposed to train a configuration on multiple worker nodes.

Bulk Synchronous Parallelism. This approach utilizes a master node and several worker nodes to conduct model selection, and MLBase [68] is a representative system. First, given an input dataset, the user specifies an ML task on the master node. Then TUPAQ, the optimizer [155] of MLBase first parses the user input, and then generates some model configurations to be evaluated. For each configuration, TUPAQ evaluates it in parallel on worker nodes because the entire training set cannot be loaded into the memory of a single machine. To this end, as shown in Fig. 5(b), TUPAQ divides the dataset into several parts, each of which is sent to a worker node. Then TUPAQ trains these partial datasets on these nodes respectively, computes the updated gradient, and sends the updates to the master node. Next, the master node computes the average updated gradient to build a new model and sends it back to worker nodes. Then the worker nodes continue to train iteratively until converge. Therefore, it has a good scalability because it can run each configuration in parallel. However, the downside is that it is harder to converge than the task parallel approach, because different parts of data are trained independently.

Parameter Server. Parameter server [84] can be taken as a fine-grained way of bulk synchronous parallelism, which also runs each configuration in parallel. As shown in Fig. 5(c), the difference is that worker nodes send the updated gradients when each mini-batch is finished and then pull the latest model from the master node. There are two classes of PS, synchronous and asynchronous parameter server. The former asks the master to compute the average updated gradient when all workers finish a mini-batch. The latter asks the master to train whenever an update comes. Therefore, asynchronous PS has a good scalability but poor reproducibility. Besides, synchronous PS has better convergence because of fresh gradient updates but higher overhead due to synchronization. Compared with bulk synchronous parallelism, PS has higher scalability and better convergence because of its finer granularity in mini-batch, but the communication cost is higher due to the frequent communication.

Model Hop Parallel. To achieve high scalability, good convergence and reproducibility, Nakandala et al. [118] proposed a model hop parallelism(MOP) framework CEREBRO, which combines the ideas of task parallel and BSP to run configurations in parallel on both task and data level. Different from the above

Methods		Scalibility	Convergence	Reproducibility
Task Parallel [116]		Low	Good	Good
Bulk Syn. Parallel [68], [155]		Medium	Medium	Good
Parameter Server(PS) [84]	Asyn. PS	Very High	Poor	Poor
	Syn. PS	High	Medium	Good
Model Hop Parallelism [118]		Very High	Good	Good

TABLE 6: Comparison of Model Selection Acceleration Methods.

works, as shown in Fig. 5(d), CEREBRO uses a decentralized architecture without the master node. Given a dataset D, a set of model configurations S to be executed and p worker nodes, it first shuffles and partitions D and different partitions will be distributed to the p worker nodes. Since there are p worker nodes, CEREBRO can evaluate p configurations in S in parallel. To this end, in each training epoch, it first selects p models from S to train on different data partitions. Then, each model only has the training result over a partition instead of the entire dataset, so these p models will *hop* to next worker node in accordance to the pre-generated *hop schedule*. After p hops, all models will be trained over the entire dataset D. Then another p models are selected to train until all models in S have been evaluated. One of advantages of CEREBRO is that for each model, the entire process is logically equivalent to sequential SGD, which results in good convergence. Furthermore, by saving the *hop schedule*, it is easy to restore the SGD sequence for every model and thus has a good reproducibility.

3.3 Acceleration of Train & Inference

Besides feature and model selection, the most important part in ML is model training and inference. In the ML community, training a model means to minimize a loss function, where a number of optimization techniques can be utilized such as gradient descent, back propagation, etc. Since there exists a lot of data to be trained (inferred) and the data operations (matrix multiplication, matrix inversion) in this step is always computationally intensive, model training and inference are always time-consuming, especially for training. Hence, in this subsection, we focus on how to leverage database techniques to accelerate model training and inference.

3.3.1 Linear Algebra Optimization.

Most ML algorithms rely on computations of linear algebra (LA) operations. However, these operations, like matrix multiplication, are always time-consuming. Therefore, accelerating the computation of LA operations is to accelerate the ML algorithm to a large extent, in both training and inference stages. In this part, we discuss how to rewrite LA operations for ML acceleration.

Optimize in Database. Cohen et al. [30] propose MAD, which executes LA operations in database so that the computation of operations are likely to be accelerated because 1) data storage and computation will be managed by DBMSs uniformly, so the time of data transformation can be saved. 2) some database techniques (e.g., index, join) can be utilized for efficiency. To this end, MAD has to represent matrices using relational tables and covert LA operations to relational algebra (RA) applied in DBMSs. For example, consider matrices R and T with dimensions $m \times l$ and $l \times n$ respectively. To compute $C = R \times T$, where $C_{ij} = \sum_{k=1}^{l} R_{ik} T_{kj}$, MAD represents the table R as `R(i,j,v)`, where `v` denotes the value of the element at `(i,j)`. Hence, C can be computed by `SQL` query:

```
SELECT R.i, T.j, SUM(R.v*T.v)
FROM R, T
WHERE R.j = T.i
GROUP BY R.j, T.i;
```

Also, MAQ supports other LA operations like add, element-wise multiplication, transposition etc. However, it works well for sparse matrices, but not suitable for dense matrices because the dense ones induce much storage overhead. To address this, MAQ allows users to design complicated functions (UDFs) to compute LA operators, e.g., the dot product of two vectors. In this way, training and inference can be accelerated in database.

Rule-based Optimization. Although MAD accelerates LA operations by executing them in DBMSs, it neglects that LA operations can be optimized for further improvement of efficiency. To be specific, rewrite or fusion of LA operators can avoid large unnecessary intermediate results. For example, if one aims to compute the operation $sum(\lambda \odot X)$, she has to first materialize the intermediate $\lambda \odot X$ matrix, which leads to extra time and storage overhead. If she rewrites it as $\lambda \odot sum(X)$, the intermediate matrix can be removed. To this end, Culumon [53] and SystemML [11] propose to optimize LA expressions using rule-based methods. The key idea is to leverage different rules to select appropriate logical and physical plans considering a cost model. Specifically, the optimizer of SystemML [14] first models the LA operators in each expression as a DAG, and then performs static analysis such as common subexpression elimination and rule-based algebraic simplification, which can be regarded as logical level optimization. Afterwards, the optimized DAG is then used for cost-based physical level operator selection, where the optimizer can decide which backend (CPU/GPU/map-reduce for distributed computing) to use by taking the size of matrices, sparsity, and memory constraint of a single machine into account.

RA-based Optimization. Hence, the rule-based methods match the patterns in a LA expression for optimization, which suffer from two limitations. First, a large number of patterns and matrices with different formats (sparse/dense) result in heavy development efforts. Second, slightly changed patterns (e.g. add ϵ to avoid dividing zero) can make existing rules inapplicable. To address this, Elgamal et al. purpose SPOOF [37] framework that can apply rewrites and fusion on LA operators automatically. Specifically, SPOOF takes as input LA operators, transforms input into a RA plan, which will be optimized by the cost model in databases, considering typical RA rules (e.g. aggregation pushdown, selection pushdown, join elimination). Finally, SPOOF transforms the optimized RA plan back to LA operators. In a nutshell, by leveraging RA, SPOOF transfers the burden of managing patterns of LA expressions from developers to the DBMS optimizer. Moreover, the optimizer can also handle slightly changed patterns.

SPORES [171] pushes the idea in SPOOF one step further with seven rules to rewrite RA expression, and proves that the rewrite rules are complete, i.e. with just seven rules, a RA expression corresponding to a LA expression can be transformed to any other equivalent RA expressions. With the completeness guarantee, SPORES can search much larger space to find the optimal equivalent expression. In order to search efficiently, SPORES adopts a compiler technique called equality saturation, which splits the optimization process into two phases: *exploration* and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

extraction. The former one builds a data structure, e-graph [36] that compactly generates and stores all equivalent expressions by applying these rules. In the following extraction step, SPORES searches the optimal result on the graph.

Besides, LARADB [54] also notices the close relationship between RA and LA, which aims at designing a specific algebra as an intermediate representation for both LA and RA. Then it rewrites the representation for optimization, and executes the corresponding physical plan in database. Moreover, motivated by the observation that operator fusion may lead to redundant computations in common subexpression elimination, Boehm et al. [13] propose a novel cost model that takes frequencies of subexpressions into account to select the best fusion plan.

3.3.2 Factorized ML

In the ML community, people always assume that the input data are stored in a single table. However, enterprise data are always scattered in multiple relational tables due to normalization, so data analysts must materialize data in a single table by joining all tables and export data out of databases. This causes large computational cost and storage overhead, as well as data leakage risk. Factorized ML is purposed to address these issues by supporting machine learning on multiple relational tables and pushing ML operators down to each single table.

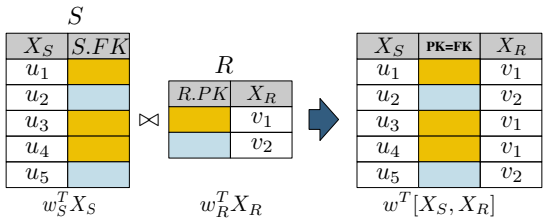

Fig. 6: A Factorized Machine Learning Example

Kumar et al. [74] purpose factorization method for generalized linear model (GLM). Given two tables S with n_S rows and R with n_R rows, a linear model on feature sets $x_S \subset S$ and $x_R \subset R$ needs to be trained. S contains a foreign key FK to join with R. When training with the gradient decent method, $\nabla F = \Sigma_{1 \le i \le n} G(y, w^T x_i) x_i$ is calculated in each iteration, where F is loss function and G is derivative function of loss on a single training instance. As shown in Fig. 6, if the model is trained on the joined data, $w^T x = w_S^T x_S + w_R^T x_R$ will be calculated $n_{S \bowtie R}$ times, while only n_S times $w_S^T x_S$ and n_R times $w_R^T x_R$ calculation are necessary when the inner product is pushed down to single table. Hence, factorized GLM pushes the inner product operation down, in order to fully utilize the property of normalized data in databases to avoid redundancy in calculation and reduce storage overhead of joined data by calculating the inner product on-the-fly.

Schleich et al. propose **F** [144], focusing on arbitrary join rather than the PK-FK one, but it is restricted to linear regression task, which designs a system to build regression model in database efficiently. Furthermore, **F** also leverages the mathematical expression rewrite strategy to further optimize the efficiency. Specifically, for linear regression task with least square loss, the gradient $\nabla F = \Sigma_{i=1}^{m} (\Sigma_{k=0}^{n} \theta_k x_k^{(i)}) x_j^{(i)}$ will be calculated in each iteration($x_k^{(i)}$ denotes the k-th feature of i-th data item in), because θ will be updated every iteration. However, if one rewrites it to $\Sigma_{k=0}^{n} (\Sigma_{i=1}^{m} x_j^{(i)} x_k^{(i)}) \theta_k$, $x_j^{(i)} x_k^{(i)}$ can be shared between iterations so that the efficiency is improved. **AC/DC** [65] further generalizes **F** to non-linear models and categorical features under functional dependencies. LMFAO [145] is a system that builds on **F** and **AC/DC** to support aggregation operations(commonly used in ML optimization) over the join. For efficient execution, LMFAO also leverages the code generation technique to support multi-output optimization, i.e., calculating multiple results in one pass of join tree traversal.

Although factorized ML methods can reduce redundancy of computation compared to traditional compute-after-join ways, they are difficult to be applied as they need complete re-implementation of ML algorithms, which takes too much effort. To address this, Chen et al. develop a framework MORPHEUS [24] that aims to support a number of ML algorithms with few modifications efficiently. To this end, MORPHEUS defines a new data type called normalized matrix that connects the high-level machine learning algorithms and low-level matrix computation operators. Then almost all ML algorithms built on basic matrix operations can be implemented by the normalized matrix. Specifically, for table S and R with S.FK-R.RID equal join, the normalized matrix for joined table is $T_N = [S, KR]$, where K is a matrix and $K[i, j] = 1$ if FK of i-th row in S is equal to j, otherwise 0. With matrix K, normalization information from origin data can be preserved, and basic matrix operations such as matrix multiplication and cross-product and matrix inversion can be implemented with pushdown. Moreover, Li et al. extends normalized matrix, and purposed MorpheusFI [86] to support efficient quadradic pairwise feature interaction i.e., including pairwise product of features to boost model accuracy. Yang et al. [179] explore the property of L_2 norm term, and figure out a method to factorize Gaussian kernel SVM with Gaussian kernel property.

3.3.3 Computation Graph Optimization

In this part, we discuss a higher level optimization, i.e. computation graph optimization in deep neural networks (DNN). Nowadays, many typical DNN architectures have become larger and deeper, resulting in a large amount of computing resources. To mitigate this, it is natural to optimize computations in a DNN, which is defined as a computation graph of mathematical operators. Existing deep learning frameworks like TensorFlow, PyTorch, optimize an input graph by performing *greedy rule-based* substitutions on it. Each substitution replaces a subgraph matching a specific pattern with a new subgraph that is equivalent to the original one. For example, operator fusion integrates multiple operators into a single one, which removes intermediate results, and thus system overheads such as memory accesses are reduced.

The main challenge of computation graph optimization is that *the search space of all equivalent computation graphs is exponentially large*, which inevitably brings huge cost and requires higher search efficiency. Existing deep learning systems solve the problem using a rule-based method that continuously improves the efficiency, which may get stuck at the local optimum.

To solve this problem, Jia et al. propose MetaFlow [59], which establishes a cost model to efficiently estimate the time usage of computing the network, considering rules with respect to the statistics like FLOPs, memory usage, and the number of kernel launches etc. Unlike Pytorch, MetaFlow considers rules with increasing cost in the intermediate computation graphs to seek more optimization opportunities. For example, as shown in Fig 7, an enlarge substitution is first applied to enlarge a convolutional layer from $1 \times 1 \times 64$ to $3 \times 3 \times 64$, which in fact increases the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2022.3148237, IEEE Transactions on Knowledge and Data Engineering

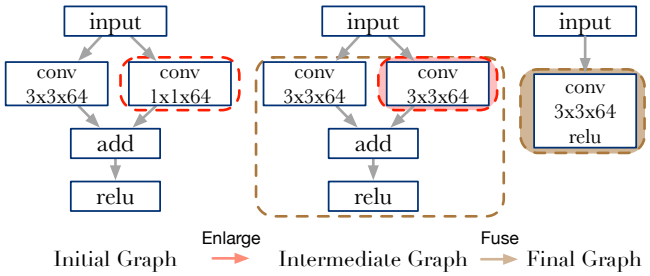

Fig. 7: A Graph Optimization Example

intermediate total cost. However, after subsequent substitutions that fuse operators like add and relu together, the final optimized computation graph can achieve a $1.3\times$ speedup. During the search process, cost-based backtracking search is utilized. Besides, to ensure search efficiency, the computation graph is divided into smaller subgraphs and optimize respectively. However, the substitution rules in MetaFlow are designed manually, which is human expensive. Therefore, Jia et al. further propose TASO [58], where an automatic rule generator is proposed. The generated rules can also be automatically verified as long as the properties of the operators, e.g. associativity, commutativity are provided. In this way, less human efforts are needed.

To thoroughly optimize the problem, Fang et al. [41] formally define it as a cost minimization problem. Specifically, taking as input the initial computation graph G, a set of substitution rules Φ, the substitution length limit K and a cost function f, the optimizing computation graph problem is to find a graph substitution sequence P with length no more than K such that the cost of the optimized graph G' after applying P is minimized. The problem is proven to be NP-hard and an efficient heuristic method with polynomial time complexity is proposed for more practical usage.

The above methods all apply substitutions in a *sequential* manner, i.e., choosing substitution rules one by one. Such methods are sensitive to the order in which the substitutions are applied and often only explore a small fragment of the exponential space of equivalent graphs. In other words, the order in which substitutions are applied directly affects the quality of the final graph. Therefore, TENSAT [180] presents a novel technique for computation graph optimization that employs a compilation technique named *equality saturation* [157], as discussed in Section 3.3.1, to apply all possible substitutions at once.

Memory optimization. Given a computation graph, there exist some memory management techniques to optimize the memory usage while executing the ML algorithms over the graph. They can help to save the memory space such that the database/ML systems can be able to run more ML tasks simultaneously.

Generally speaking, the approaches of memory management techniques, especially for GPU memory, can be mainly categorized into 4 groups. (1) *Model compression* saves the memory of deep learning algorithms by reducing the precision of parameters or simplifying the complex neural architectures. Typically, mixed precision numbers, single bit and half precision can be utilized in the deep neural networks to save memory [31], [76], [107], but they may introduce quantization errors [158]. Also, model pruning methods can also be applied to shrink the complexity of the networks [50], [48]. However, the above methods may decrease the accuracy, so there is a trade-off between the model compression and performance. (2) *Model sharing* aims to reduce the memory usage by sharing, which can be classified into two categories. One is to use the in-place operation that stores the output of a layer exactly at the physical address of the input. For example, given a layer $y = relu(x)$, y can be directly stored at the place of x. The other one is buffer reuse [26], [25], which shares the memory among variables existing in different lifetimes without overlapping. Once the computation graph is built, we can capture the data dependencies so as to provide smarter memory allocation. (3) *Computation trading* refers to freeing some intermediate results (*e.g.,* feature maps) while forward propagation, so as to save the memory of graph computation. However, during backward propagation, these results have to be recomputed to derive the gradients. Therefore, existing works [26], [168], [125] focus on judiciously selecting the intermediate results that are easy to compute to drop. (4) *Memory swapping* aims to swap variables that are not being used to CPU, and then back to the GPU memory before their following access. Most works [168], [34] rely on the user to specify which layers or tensors should be swapped, and thus an automatic approach to memory swapping is desired.

3.3.4 Hardware Acceleration

Hardwares are always utilized to accelerate ML algorithms, like FPGA, GPU. Recently, many researchers have studied how to use specialized hardware design to accelerate the in-RDBMS execution of complicated data analytics.

Mahajan et al. propose DAnA [105], a system that uses FPGA to conduct acceleration in databases. There are mainly two contributions. On the one hand, DAnA is an easy-to-use system that encapsulates the complexity of execution of FPGA and DB using UDFs. On the other hand, the limited CPU memory bandwidth limits the potential of FPGA if they are connected together directly. To address this, DAnA proposes *Striders* that bypass CPU and its memory subsystem, directly communicating with the RDBMS buffer pool. Besides, to fully utilize the FPGA bandwidth, the data are transferred to FPGAs at a granularity of pages using multi-threads.

ColumnML [63], on the other hand, focuses on FPGA acceleration on column-store data frequently used in OLAP database. It is not trivial to integrate ML and OLAP database efficiently because they are different in the execution model and data layout, and ColumnML turns to FPGA for help. Specifically, ColumnML concentrates on stochastic-coordinate-descent(SCD) over data in columnar format. Directly using basic SCD algorithm suffers from inefficiency because (1) original SCD algorithm is poor in cache locality because it has to keep a huge amount of intermediate state; (2) in column-stores, data are often compressed and encrypted [7], and thus recover the data can be costly, which can be well accelerated by FPGA. For the first challenge, ColumnML proposes a partitioned SCD (pSCD) algorithm that divides the complete dataset into partitions. SCD is then performed on each partition independently and the final model is collected by model average. With pSCD, the complexity of memory accesses can be reduced due to cache locality. Besides, pSCD is more parallelizable, requiring little synchronization and can be well accelerated by FPGA. In this way, this process can fully utilize the available memory bandwidth of FPGA and get higher throughput. With respect to the second challenge, ColumnML improves also by leveraging the architectural flexibility of FPGA. By making the decompression and decryption in parallel in the whole pipeline using FPGA, the data transformation can be completed efficiently without any throughput reduction.

There have also been works on GPU-based database systems. Previous works [44], [83], [167], [182] have demonstrated that these GPU-based systems can bring efficiency improvement on analytical workloads. This improvement comes from that in-memory analytics is typically memory bandwidth bound, and GPU generally has larger bandwidth (about 10×).

However, the above GPU-based database systems only treat GPU as a coprocessor. For any query, correlated data are shipped between CPU and GPU over PCIe. PCIe is an order of magnitude slower than GPU, and can even be slower than CPU. Therefore, previous coprocessor design is suboptimal and does not fully utilize the ability of GPU. To this end, Anil et al. propose Crystal [152] that avoids the process of such data transfer by implementing database operator completely on GPUs, which induces several challenges because of the main difference between CPU and GPU, i.e. GPU has a larger number of weaker cores.

Firstly, synchronization between a large number of threads can be costly. Secondly, each thread of GPU has much smaller memory and weaker computation ability, which limits the operator implementation. Crystal proposes a *Tile-based* execution model to solve these challenges. GPU threads are generally grouped into thread blocks that are defined as *tiles*. Within each *tile*, threads can communicate through a larger shared memory. Therefore, using *tile* as the basic unit both reduces the fetch operations to the global memory and brings more available memory to each single thread. In the paper, operators including projection, selection and hash join designed for GPU implementation are proposed, with which Crystal avoids data transfer over PCIe and achieves higher efficiency. However, since Crystal needs the memory of GPU can hold the complete dataset, current supported dataset size is limited and generally requires higher financial cost.

4 Post-ML: Model Management

While training the model, developers usually try dozens of model architectures, tune the hyper-parameters, and then check their performance. Therefore, how to manage the trained models becomes a challenge, which aims to store, version, query, deploy, monitor, and debug the ML models (including managing their metadata). In this section, we discuss the challenges of model management from the management perspective and survey some works that aim to improve the effectiveness and efficiency of model management.

4.1 Model Storage, Versioning, and Query

After training, multiple models are built. In this section, we study model storage, versioning and querying, which have close relationships with each other. First, these models should naturally be stored for further analysis. Then for these stored models, version control should be applied to track them for flexible search, i.e., model querying. Also, declarative languages can also be designed to achieve user-friendly querying.

ModelDB [161] is the first open-source model management system that aims to automatically track, index, and explore ML models, which consists of three components: native client libraries for supporting different ML environments, a backend for storing the model, and a web-based visualization interface for exploring and analyzing the model metadata. Specifically, the native client libraries component is designed for automatically managing models in their native environments (*e.g.,* spark.ml, scikit-learn), so as to minimize the usage cost for new users (*e.g.,* changing their familiar ML environment). Next, the backend component will store the ML pipelines as a sequence of actions, which are stored in a relational database in the backend. The ML models are stored and indexed using a customized storage engine. Finally, ModelDB designs an easy-to-use web-based interface for the users to explore and analyze a large number of models and pipelines. With the help of visual interface, the users can easily review, compare, and analyze the models and pipelines.

Similar to ModelDB, ModelHub [110] is an end-to-end model lifecycle management system that enables users to store, version, snapshot, query, and reuse models and their data artifacts (*e.g.,* hyperparameters, trained snapshots), but it especially optimizes for the deep learning workflow. ModelHub follows the client-server architecture, with three key components: a model versioning system, a domain-specific language module for assisting developers, and a hosted deep learning model sharing module.

4.2 Model Diagnosis

Given a trained model, its performance always does not achieve the requirement. Hence, the goal of model diagnosis is to assist ML developers to understand why the training process doesn't achieve an acceptable performance and help developers find the root cause in model or data, so as to make efforts to improve the performance. To this end, some developers analyze trained models (and data artifacts) by manually looking into the model in an ad-hoc, one-off basis, e.g., writing a python script to analyze the embeddings vector in deep neural networks. However, the above method faces two main challenges w.r.t. data management, *i.e.,* (1) large storage overhead because of a large number of data artifacts produced during the model training and testing stages; (2) high computational cost because diagnosing among the large number of data is not efficient.

To address the aforementioned challenges, there are a line of systems [27], [51], [108], [109], [146], [160] proposed by the data management community. Generally speaking, such systems devise a variety of storage optimization techniques such as de-duplication and quantization to reduce the storage size or develop sampling-based techniques to speed up the model diagnosis query process.

MISTIQUE [160] aims to efficiently capture, store, and query model intermediates (*i.e.,* data artifacts related to the model) for model diagnosis. It consists of three steps: (1) Model Intermediates Logging: it allows developers to call APIs to request MISTIQUE for logging model intermediates for each model layer. (2) Access Data Artifacts: next, developers can access the data artifacts logged in MISTIQUE via the APIs. The data will be returned as the Python Numpy arrays. (3) Run Model Diagnostic Query: then, developers can perform model diagnostic queries on these data artifacts, or use predefined analytic functions for analysis. The key technical contribution among the three steps is how to store the large volume of the model intermediates, where three key ideas are adopted. First, for deep neural networks, MISTIQUE adapts *activation quantization* techniques to reduce the storage cost. To be specific, it approximates the neural networks using floating-point number, and thus it can dramatically reduce the storage cost without sacrificing much accuracy; Second, it devises a *similarity-based compression* mechanism to remove redundant data between model intermediates both for ML pipeline. Finally, MISTIQUE proposes *adaptive materialization* to materialize those intermediates visited frequently, *e.g.,* the prediction part of a model. However, materialization also induces the storage cost. Thus, it adopts a cost model to trade-off the increase in storage against the improvement of query efficiency.

Besides diagnosing the performance of a ML model, understanding and interpreting the reasons behind the performance is also important. DeepEverest [51] is a system that focuses on accelerating top-k *interpretation by example* queries [89], *e.g.,* retrieving the top-k maximally activated neuron. For example, given a trained DNN for an image classification task, a developer might be interested in understanding which parts of an image makes the DNN predicting correctly. To this end, the developer has to inspect which groups of activated neurons act as the semantic detectors of features in the image (*e.g.,* ears). To avoid materializing as many activation values as possible, DeepEverest builds an index, called *Neural Partition Index*, on the query search keys. Then, it devises a query execution algorithm, named *Neural Threshold Algorithm*, to utilize the *Neural Partition Index* to progressively process those inputs that potentially produce the top-k results needed by the users. Benefiting from the above optimization techniques, the query execution time can be significantly reduced by reducing the number of inferences performed by DNN, while guaranteeing the precision of the top-k results.

DeepBase [146], [27] provides a unified interface for developers to express their analysis to quickly understand the neural network behavior. It abstracts the model diagnostic queries as hypotheses verification task. More specifically, DeepBase takes as input a testing dataset, a trained neural network model, a set of Python codes that contains hypotheses of what the model may be learning (it denotes as *hypothesis functions*, and a scoring function (*e.g.,* a measure of statistical dependency). Next, the system computes a set of scores that measure the affinity between the hypotheses and the neural network models' hidden units. Similar to other model diagnosis systems, the efficiency bottleneck of DeepBase is the large volume of activations to be extracted, stored, and matched according to the given hypotheses. To alleviate these issues, DeepBase proposes three main optimization techniques, *i.e.,* cacheing, early stopping and streaming execution.

4.3 Model Deployment and Serving

In real-world ML applications, deploying and serving ML modes can be challenging because: (1) it usually needs to incorporate multiple evolving modules together (*e.g.,* integrating TensorFlow into software infrastructure) for prediction in the production environment, but ML developers may be unfamiliar with such operations. (2) the real-time data (*e.g.,* IoT data) for prediction comes in a fast stream way and some applications (*e.g.,* finance) have stringent latency requirements. Thus the model deployment and serving infrastructure should have: (1) easy-to-use systems for developers to deploy their models seamlessly; and (2) high throughput, low latency, and good performance.

Clipper [32] is a general-purpose model serving system that provides low-latency and high throughput prediction services, which consists of two layers: the model selection layer and the model abstraction layer. When ML applications send the prediction requests to the Clipper, the model selection layer first analyzes the requests and then dispatches the requests to one or more models through the model abstraction layer. The reason why it dispatches the prediction requests is because it allows many candidate models to work together and combines their prediction results to boost application accuracy. After the model abstraction layer receiving the requests from the model selection layer, the model abstraction layer calls target models across machine learning frameworks (*e.g.,* TensorFlow and Scikit-Learn) via APIs. Clipper devises a set of optimization techniques (*e.g.,* caches and batching) in the model abstraction layer to ensure low latency and high throughput and adapts bandit and ensemble approaches in the model selection layer to improve the accuracy of prediction results and estimate the uncertainty.

TensorFlow-Serving [120], a flexible and high-performance model serving system for ML system, is developed almost concurrently with the Clipper. TensorFlow-Serving shares some similar features with the Clipper, *e.g.,* both are dedicated to a general-purpose model serving system that is model-agnostic and has similar optimization techniques such as batching. TensorFlow-Serving is powered by the state-of-the-art production infrastructure used by Google, while Clipper is a research-oriented prototype to demonstrate its research idea. TensorFlow-Serving supports off-the-shelf integration with TensorFlow models, and can be extended to serve other types of models with a little extra effort.

4.4 ML Pipeline Debugging

ML pipeline debugging, which is usually in the production environment, refers to debugging unexpected results under new testing data and finding root causes of them. It shares many common features with ML model diagnosis (Section 4.2). For example, both need to manage a large volume of data artifacts and model instances, helping the user to verify their hypotheses. However, ML pipeline debugging focuses on debugging when new testing data does not perform well after deployment, while model diagnosis debugs models while developing.

Generally speaking, the reasons of failures in an ML pipeline are two-fold: (1) data errors: the issues of the input data or the intermediate data produced by the code, and (2) model bugs: a set of unsuitable settings in the model, including raw codes, hyper-parameters, computational modules, etc. Identifying the reasons of a failed ML pipeline is challenging, often requiring much time and human efforts, but is still error-prone. In the data management community, there are a line of works focusing on the above two problems of ML pipeline debugging, *i.e.,* data debugging [173], [136], [170], [159] and model debugging [94], [96], [95].

In some cases, although the query (*e.g.,* a SQL query) and queried data (*e.g.,* the customer data) are correct, the errors in the training data may cause the trained model to make false predictions and further lead to wrong outputs. Given this context, Rain [173], a training data debugging system, helps the user capture training data errors by leveraging not only the ML model and data but also user complaints (*i.e.,* those errors annotated by the user) about final or intermediate outputs. To address the training data issues, Rain aims to find a minimum set of training records such that if they are removed, the user complaint could be solved. Another challenge for debugging training data is that if we remove records in the training set, we need to retrain the model to estimate how the model parameters change. Obviously, it is time-consuming to retrain the model to estimate the influence of removing some training records. To alleviate this issue, Rain leverages the *influence functions* [67] to estimate how the model parameters change by removing a set of training records and without retraining the model.

Data X-Ray [170] and Dagger [136] are also the data debugging systems. Data X-Ray [170] focuses on debugging a large-scale data pipeline by explaining where and how errors appear in the data generation process. Data X-Ray treats data debugging as the problem of finding shared features among erroneous data elements. It takes as input the provenance of pipeline instances

and annotated errors and outputs a set of features for explaining the reasons for annotated errors. Similarly, Dagger [136] also can identify data issues in the data generation process, but it typically focuses on diagnosing the issues of intermediate data produced in the pipeline. Flipper [159], sharing many common features with the above two, is a human-in-the-loop framework for debugging the training data produced by generative models. It provides users with easy-to-interpret high-level descriptions of issues in the training data and then helps users to improve the quality of their training data.

The ML pipeline may crash when trying a set of new parameter values, it is time-consuming and tedious for developers to valid new ML pipeline instances and find the root cause of failures. Thus, MLDebugger [96] is a model debugging system that *automatically and interactively* captures a set of root causes for failures in the ML pipeline. It first takes a group of ML pipeline instances as input, then derives a hypothesis about possible root causes. Next, it devises an algorithm to wisely choose new ML pipeline instances to run untested parameter values. MLDebugger repeats the above steps until the time budget is consumed up or a definitive root cause is derived. Finally, the system returns a set of concise explanations for the possible root cause. Thus, the user can understand the debugging results and interpret the possible reasons for failures in the ML pipeline, and then act on them. BugDoc [95], [94], extending from the MLDebugger, is designed to identify the abnormal behavior in *general computational pipelines* that may be due to errors in data, codes, or other actions in the pipeline.

Apart from the above works, there are also many papers that investigate ML pipeline debugging in the ML and software community, we refer interested readers to recent works and workshops [3], [154], [122] in these communities.

5 Research Challenges and Opportunities

In this section, we will discuss some research challenges that are also relevant to data management problems on machine learning, but are not yet well studied.

5.1 Data Preparation

Tuple-level data discovery for ML. Existing tuple-level data discovery approaches [188], [153] mainly focus on enriching the dataset without considering the impact on downstream tasks. Besides, although active leaning based methods [147] directly benefit ML, they need expensive human label. Other methods in the field of computer vision [153] just apply augmentation operations on original data to generate new data. However, existing data resources (e.g., data market, data lake) provide opportunities to discover fresh training data from heterogeneous data pools. The challenge lies in how to judiciously select beneficial data for ML from heterogeneous resources.

More general data cleaning for ML. Existing methods [70], [64], [85], [69] of cleaning for ML mainly focus on cleaning a certain type of dirty data (e.g., missing values) for a certain type of ML model, especially the traditional model (e.g., SVM) rather than deep learning. Therefore, there is a lack of more general methods that can judiciously select to clean the data with all possible dirty types, with the goal of improving an arbitrary type of model. The challenge is how to measure the influence of different dirty types on models, especially the deep learning model.

Theoretical guarantee for data labeling. As we know, model performance close relationship with the labeling quality [82], but we do not know the exact relation theoretically. To be specific, given a model performance requirement, if we can deduce which part of data needed to be labeled and whether it is acceptable to use weak labels, fine-grained and cost-effective data labeling can be achieved. However, it is very challenging because the training process between the data and final model is complicated.

Efficiency of data preparation. As a consensus, data preparation takes 80% time in a data science pipeline [2], so it is significant to improve the efficiency. First, data discovery is always time-consuming because it always incorporates complex join operations [28], [91]. Hence we can accelerate this process by some heuristic solution like sampling. Data cleaning and labeling are inefficient because they always rely on humans to solve the problem [82], so it is necessary to utilize more automatic approaches in the pipeline. Another interesting direction is to take them as a whole and remove some unnecessary, inefficient steps, considering the data characteristics and model performance.

5.2 Model Training & Inference

End-to-end ML optimizer in DB. In this survey, we show how to use DB techniques to optimize several steps in ML, including feature selection [75], [151], [79], [174], model selection [116], [68], [118], training and inference [30], [53]. But these methods optimize separately without considering that they can work together to achieve more optimization. It is challenging and significant to conduct an end-to-end ML optimizer.

In-database deep learning. It is very challenging to support ML inside the database, including feature engineering, model selection and training. Existing works [30], [74] discussed in this survey mainly focus on learning traditional ML models (e.g., logistic regression) in databases. But for deep learning, the complicated neural architectures and a number of parameters computation make it difficult to train in database. Besides, it is also challenging to fully integrate GPU into database for efficient deep learning.

Hybrid DB&ML optimizer. Nowadays, both DB and ML operators are needed in most real-world applications, so an end-to-end hybrid ML&DB system is necessary, where a customized operator is one of the most important modules. There exist many approaches [163], [181], [183], [190], [191] to optimizing the DB operators. For the hybrid optimizer, the execution order of ML and DB operators matters, where novel cardinality/cost estimation strategies can be designed. Besides, the ML operator is likely to incorporate multiple tables (join), how to coordinate this type of join and the join of DB queries is a challenge. Moreover, a customized declarative language and appropriate storage engine should be designed.

Robust learning for systems. Existing ML algorithms do not consider fault tolerance in systems. Especially in a distributed environment, the failure of a process is likely to make the whole task crash. Therefore, we can leverage the error tolerance approaches of database systems to improve the robustness of in-DB learning. To guarantee the business continuity under predictable and unpredictable disasters, ML systems must have fault tolerance and disaster recovery capabilities.

5.3 Model Management

Streaming model management. Model training is an iterative process, and a large number of parameters are generated along with the training process and should be managed carefully. Existing methods like [161] take these parameters as static information,

but in fact, they can be taken as the streaming data. For example, we can study how to store these models using a light overhead. Besides, whether multiple models generated during training can be integrated for more accurate inference. What's more, there may be some outliers in the streaming data that can provide beneficial hints for performance improvement.

Model decomposition and reconstruction. Multiple models are often built for an ML task. For example, given an image classification task, many pre-trained models can be utilized [129]. In this situation, one may ask that whether it is feasible to automatically decompose each model to multiple blocks, and then reconstruct from these blocks of different models for the specific ML task. The challenges lie in which granularity to decompose the model, how to select proper decomposed blocks for the performance of the ML task and how to conduct this process efficiently.

Systematic and collaborative ML pipeline debugging. Current practices for ML pipeline debugging usually separate data and model debugging [27], [51], [108], and often debug in a serial pipeline. In this way, some errors in certain steps may propagate. Thus, how to debug the ML pipeline in a holistic way is challenging. On the other hand, ML pipelines involve participants of different roles, including ML/data scientists, developers, operation and maintenance staff, and customers. They have different rights, expertise, and perspective for the ML pipeline. For example, the customers may complain about and report the unexpected results, and the operation staffs deliver the issues to the developers to find root causes and fix them. Thus, how to enable collaborative ML pipeline debugging is important in the production environment.

6 Conclusion

In this paper, we summarize the recent techniques on data management for ML from three aspects. The first is data preparation, including data discovery, data cleaning and data labeling. The second one is the acceleration during ML training and inference. The last one comprises steps after ML models are built, including model storage & query, model diagnose, deployment and pipeline debugging. We also propose some research challenges and open problems in this field.

Acknowledgement. This work is supported by NSF of China (62102215, 61925205, 61632016), Huawei, TAL education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation(2021M691784), Shuimu Tsinghua Scholar and Zhejiang Labs International Talent Fund for Young Professionals.

References

[1] Amazon mechanical turk. https://www.mturk.com/.
[2] Cleaning big data: Most time-consuming, least enjoyable data science task, survey says. https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=2097de186f63.
[3] Debugging machine learning models, iclr 2019 workshop. https://debug-ml-iclr2019.github.io/.
[4] *Recommender Systems Handbook*. Springer, 2015.
[5] *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, 2019.
[6] M. R. Anderson and M. J. Cafarella. Input selection for fast feature engineering. In *ICDE 2016*, pages 577–588.
[7] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. In *CIDR 2013*. www.cidrdb.org.
[8] K. Atarashi, S. Oyama, and M. Kurihara. Semi-supervised learning from crowds using deep generative models. In *AAAI 2018*, pages 1555–1562.
[9] S. H. Bach, B. D. He, A. Ratner, and C. Ré. Learning the structure of generative models without labeled data. In *ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*. PMLR, 2017.
[10] B. Boecking, W. Neiswanger, E. P. Xing, and A. Dubrawski. Interactive weak supervision: Learning useful heuristics for data labeling. *CoRR*, abs/2012.06046, 2020.
[11] M. Boehm and M. Dusenberry. Systemml: Declarative machine learning on spark. *Proc. VLDB Endow.*, 9(13):1425–1436, 2016.
[12] M. Boehm, A. Kumar, and J. Yang. Data management in machine learning systems. *Synthesis Lectures on Data Management*, 11(1), 2019.
[13] M. Boehm, B. Reinwald, D. Hutchison, P. Sen, A. V. Evfimievski, and N. Pansare. On optimizing operator fusion plans for large-scale machine learning in systemml. *Proc. VLDB Endow.*, 11(12):1755–1768, 2018.
[14] M. Böhm and D. R. Burdick. Systemml's optimizer: Plan generation for large-scale machine learning programs. *IEEE Data Eng. Bull.*, 37(3):52–62, 2014.
[15] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101, 2009.
[16] C. Chai, L. Cao, G. Li, J. Li, Y. Luo, and S. Madden. Human-in-the-loop outlier detection. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *SIGMOD 2020*, pages 19–33.
[17] C. Chai, J. Fan, and G. Li. Incentive-based entity collection using crowdsourcing. In *IEEE 2018*, pages 341–352.
[18] C. Chai, J. Fan, G. Li, J. Wang, and Y. Zheng. Crowdsourcing database systems: Overview and challenges. In *IEEE 2019*, pages 2052–2055.
[19] C. Chai, J. Fan, G. Li, J. Wang, and Y. Zheng. Crowd-powered data mining. *CoRR*, abs/1806.04968, 2018.
[20] C. Chai, G. Li, J. Fan, and Y. Luo. Crowdsourcing-based data extraction from visualization charts. In *ICDE 2020*, pages 1814–1817.
[21] C. Chai, G. Li, J. Fan, and Y. Luo. Crowdchart: Crowdsourced data extraction from visualization charts. *IEEE Trans. Knowl. Data Eng.*, 33(11):3537–3549, 2021.
[22] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *SIGMOD 2016*.
[23] C. Chai, G. Li, J. Li, D. Deng, and J. Feng. A partial-order-based framework for cost-effective crowdsourced entity resolution. *VLDB J.*, 27(6):745–770, 2018.
[24] L. Chen, A. Kumar, J. F. Naughton, and J. M. Patel. Towards linear algebra over normalized data. *Proc. VLDB Endow.*, 10(11), 2017.
[25] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.
[26] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.
[27] Y. Chen, Y. Shi, B. Chen, T. Sellam, C. Vondrick, and E. Wu. Deep neural inspection using deepbase. In *NIPS LearnSys Workshop*, 2018.
[28] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger. ARDA: automatic relational data augmentation for machine learning. *Proc. VLDB Endow.*, 13(9):1373–1387, 2020.
[29] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD Conference 2015*, pages 1247–1261.
[30] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. MAD skills: New analysis practices for big data. *Proc. VLDB Endow.*, 2(2):1481–1492, 2009.
[31] M. Courbariaux, Y. Bengio, and J. David. Low precision arithmetic for deep learning. In Y. Bengio and Y. LeCun, editors, *ICLR 2015*.
[32] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *NSDI 2017)*, pages 613–627.
[33] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
[34] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. Geeps: scalable deep learning on distributed gpus with a gpu-specialized parameter server. In C. Cadar, P. R. Pietzuch, K. Keeton, and R. Rodrigues, editors, *EuroSys 2016*.
[35] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW 2012*, pages 469–478.
[36] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[37] T. Elgamal, S. Luo, M. Boehm, A. V. Evfimievski, S. Tatikonda, B. Reinwald, and P. Sen. SPOOF: sum-product optimization and operator fusion for large-scale machine learning. In *CIDR 2017*.

[38] T. Elsken, J. H. Metzen, F. Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[39] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE 2014*, pages 976–987.

[40] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[41] J. Fang, Y. Shen, Y. Wang, and L. Chen. Optimizing DNN computation graph using graph substitutions. *Proc. VLDB Endow.*, 13(11), 2020.

[42] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE 2018*, pages 1001–1012.

[43] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE 2018*, pages 989–1000.

[44] H. Funke, S. Breß, S. Noll, V. Markl, and J. Teubner. Pipelined query processing in coprocessor environments. In *SIGMOD Conference 2018*, pages 1603–1618.

[45] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[46] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *SIGMOD Conference 2016*, pages 795–806.

[47] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Managing google's data lake: an overview of the goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.

[48] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: efficient inference engine on compressed deep neural network. In *ISCA 2016*, pages 243–254.

[49] S. Hao, C. Chai, G. Li, N. Tang, N. Wang, and X. Yu. Outdated fact detection in knowledge bases. In *ICDE 2020*, pages 1890–1893.

[50] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *NIPS 1992]*, pages 164–171.

[51] D. He, M. Daum, and M. Balazinska. Deepeverest: Accelerating declarative top-k queries for deep neural network interpretation [technical report]. *arXiv preprint arXiv:2104.02234*, 2021.

[52] X. He, K. Zhao, and X. Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[53] B. Huang, S. Babu, and J. Yang. Cumulon: optimizing statistical data analysis in the cloud. In *SIGMOD 2013*, pages 1–12. ACM, 2013.

[54] D. Hutchison, B. Howe, and D. Suciu. Laradb: A minimalist kernel for linear and relational algebra computation. In *BeyondMR@SIGMOD 2017*, pages 2:1–2:10.

[55] I. F. Ilyas and X. Chu. *Data Cleaning*. ACM, 2019.

[56] N. Indurkhya and F. J. Damerau. *Handbook of natural language processing*, volume 2. CRC Press, 2010.

[57] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[58] Z. Jia, O. Padon, J. J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *SOSP 2019*. ACM, 2019.

[59] Z. Jia, J. J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken. Optimizing DNN computation with relaxed graph substitutions. In *MLSys 2019*. mlsys.org.

[60] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *NIPS 2013*, pages 315–323.

[61] H. Kajino, Y. Tsuboi, and H. Kashima. Clustering crowds. In *AAAI 2013*.

[62] H. Kajino, Y. Tsuboi, and H. Kashima. A convex formulation for learning from crowds. In *AAAI 2012*, volume 26.

[63] K. Kara, K. Eguro, C. Zhang, and G. Alonso. Columnml: Column-store machine learning with on-the-fly data transformation. *Proc. VLDB Endow.*, 12(4):348–361, 2018.

[64] B. Karlas, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *CoRR*, abs/2005.05117, 2020.

[65] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. AC/DC: in-database learning thunderstruck. In *DEEM@SIGMOD 2018*.

[66] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *ICLR 2015*.

[67] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[68] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR 2013*.

[69] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu. Boostclean: Automated error detection and repair for machine learning. *CoRR*, abs/1711.01299, 2017.

[70] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, 2016.

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS 2012*.

[72] A. Kumar, M. Boehm, and J. Yang. Data management in machine learning: Challenges, techniques, and systems. In *SIGMOD 2017*, pages 1717–1722.

[73] A. Kumar, R. McCann, J. F. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Rec.*, 44(4):17–22, 2015.

[74] A. Kumar, J. F. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *SIGMOD Conference 2015*, pages 1969–1984.

[75] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD Conference 2016*, pages 19–34.

[76] S. Le Grand, A. W. Götz, and R. C. Walker. Spfp: Speed without compromisea mixed precision model for gpu accelerated molecular dynamics simulations. *Computer Physics Communications*, 184(2).

[77] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[78] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR 1994*, pages 3–12.

[79] H. R. Lewis. Computers and intractability. a guide to the theory of np-completeness, 1983.

[80] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: optimizing queries with crowd-based selections and joins. In S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, editors, *SIGMOD 2017*, pages 1463–1478.

[81] G. Li, C. Chai, J. Fan, X. Weng, J. Li, Y. Zheng, Y. Li, X. Yu, X. Zhang, and H. Yuan. CDB: A crowd-powered database system. *Proc. VLDB Endow.*, 11(12):1926–1929, 2018.

[82] G. Li, J. Wang, Y. Zheng, and M. J. Franklin. Crowdsourced data management: A survey. *IEEE Trans. Knowl. Data Eng.*, 28(9):2296–2319, 2016.

[83] J. Li, H. Tseng, C. Lin, Y. Papakonstantinou, and S. Swanson. Hippogriffdb: Balancing I/O and GPU bandwidth in big data analytics. *Proc. VLDB Endow.*, 9(14):1647–1658, 2016.

[84] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI14)*, 2014.

[85] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang. Cleanml: A benchmark for joint data cleaning and machine learning [experiments and analysis]. *CoRR*, abs/1904.09483, 2019.

[86] S. Li, L. Chen, and A. Kumar. Enabling and optimizing non-linear feature interactions in factorized linear algebra. In *SIGMOD Conference 2019*, pages 1571–1588.

[87] Y. Li, G. Hu, Y. Wang, T. M. Hospedales, N. M. Robertson, and Y. Yang. DADA: differentiable automatic data augmentation. *CoRR*, abs/2003.03780, 2020.

[88] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast autoaugment. In *NeurIPS 2019*, pages 6662–6672.

[89] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[90] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *ICDM 2008*, pages 413–422. IEEE, 2008.

[91] J. Liu, F. Zhu, C. Chai, Y. Luo, and N. Tang. Automatic data acquisition for deep learning. *Proc. VLDB Endow.*, 14(12):2739–2742, 2021.

[92] T. Liu, J. Fan, Y. Luo, N. Tang, G. Li, and X. Du. Adaptive data augmentation for supervised learning over missing data. *Proceedings of the VLDB Endowment*, 14(7):1202–1214, 2021.

[93] T. Liu, J. Yang, J. Fan, Z. Wei, G. Li, and X. Du. Crowdgame: A game-based crowdsourcing system for cost-effective data labeling. In *SIGMOD 2019*, pages 1957–1960. ACM, 2019.

[94] R. Lourenço, J. Freire, and D. Shasha. Bugdoc: A system for debugging computational pipelines. SIGMOD '20, page 27332736, New York, NY, USA, 2020. Association for Computing Machinery.

[95] R. Lourenço, J. Freire, and D. Shasha. Bugdoc: Algorithms to debug computational processes. In *SIGMOD 2020*, pages 463–478.

[96] R. Lourenço, J. Freire, and D. Shasha. Debugging machine learning pipelines. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, pages 1–10, 2019.

[97] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Interactive cleaning for progressive visualization through composite questions. In *IEEE 2020*, pages 733–744.

[98] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Visclean: Interactive cleaning for progressive visualization. *Proc. VLDB Endow.*, 2020.

[99] Y. Luo and et al. Empowering natural language to visualization neural translation using synthesized benchmarks. In *VIS*, 2021.

[100] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li. Steerable self-driving data visualization. *IEEE Trans. Knowl. Data Eng.*, 34(1):475–490, 2022.

[101] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *IEEE 2018*, pages 101–112.

[102] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In G. Li, Z. Li, S. Idreos, and D. Srivastava, editors, *SIGMOD 2021*, pages 1235–1247.

[103] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Trans. Vis. Comput. Graph.*, 28(1):217–226, 2022.

[104] F. Ma, Y. Li, Q. Li, M. Qiu, J. Gao, S. Zhi, L. Su, B. Zhao, H. Ji, and J. Han. Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation. In *SIGKDD 2015*, pages 745–754.

[105] D. Mahajan, J. K. Kim, J. Sacks, A. Ian, A. Kumar, and H. Esmaeilzadeh. In-rdbms hardware acceleration of advanced analytics. *Proc. VLDB Endow.*, 11(11):1317–1331, 2018.

[106] E. K. Mallory, M. de Rochemonteix, A. Ratner, A. Acharya, C. Ré, R. A. Bright, and R. B. Altman. Extracting chemical reactions from text using snorkel. *BMC Bioinform.*, 21(1):217, 2020.

[107] M. D. McDonnell. Training wide residual networks for deployment using a single bit for each weight. In *ICLR 2018*. OpenReview.net.

[108] P. Mehta, S. Portillo, M. Balazinska, and A. Connolly. Sampling for deep learning model diagnosis (technical report), 2020.

[109] P. Mehta, S. Portillo, M. Balazinska, and A. Connolly. Toward sampling for deep learning model diagnosis. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1910–1913, 2020.

[110] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.

[111] B. Mirzasoleiman, J. A. Bilmes, and J. Leskovec. Coresets for data-efficient training of machine learning models. In *ICML 2020*, volume 119, pages 6950–6960. PMLR, 2020.

[112] B. Mirzasoleiman, K. Cao, and J. Leskovec. Coresets for robust training of deep neural networks against noisy labels. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *NeurIPS 2020*.

[113] T. M. Mitchell and W. W. Cohen. Never-ending learning. In *AAAI 2015*.

[114] S. Mittal and S. Vaishay. A survey of techniques for optimizing deep learning on gpus. *J. Syst. Archit.*, 99, 2019.

[115] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[116] P. Moritz and R. Nishihara. Ray: A distributed framework for emerging AI applications. In *OSDI 2018*, pages 561–577.

[117] A. Naghizadeh, M. Abavisani, and D. N. Metaxas. Greedy autoaugment. *Pattern Recognit. Lett.*, 138:624–630, 2020.

[118] S. Nakandala, Y. Zhang, and A. Kumar. Cerebro: Efficient and reproducible model selection on deep learning systems. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, pages 1–4, 2019.

[119] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *Proc. VLDB Endow.*, 11(7):813–825, 2018.

[120] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[121] H. Park and J. Widom. Crowdfill: collecting structured data from the crowd. In *SIGMOD 2014*, pages 577–588.

[122] J. F. Pimentel, J. Freire, L. Murta, and V. Braganholo. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Computing Surveys (CSUR)*, 52(3):1–38, 2019.

[123] E. A. Platanios, M. Al-Shedivat, E. P. Xing, and T. M. Mitchell. Learning from imperfect annotations. *CoRR*, abs/2004.03473, 2020.

[124] E. A. Platanios, A. Dubey, and T. M. Mitchell. Estimating accuracy from unlabeled data: A bayesian approach. In *ICML 2016*.

[125] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger. Memory-efficient implementation of densenets. *CoRR*, abs/1707.06990, 2017.

[126] X. Qin, C. Chai, Y. Luo, N. Tang, and G. Li. Interactively discovering and ranking desired tuples without writing SQL queries. In *SIGMOD 2020*, pages 2745–2748.

[127] X. Qin, C. Chai, Y. Luo, T. Zhao, N. Tang, G. Li, J. Feng, X. Yu, and M. Ouzzani. Ranking desired tuples by database exploration. In *IEEE 2021*, pages 1973–1978.

[128] X. Qin, Y. Luo, N. Tang, and G. Li. Making data visualization more efficient and effective: a survey. *VLDB J.*, 29(1):93–117, 2020.

[129] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pages 1–26, 2020.

[130] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282, 2017.

[131] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré. Snorkel: rapid training data creation with weak supervision. *VLDB J.*, 29(2-3):709–730, 2020.

[132] A. Ratner, B. Hancock, J. Dunnmon, R. E. Goldman, and C. Ré. Snorkel metal: Weak supervision for multi-task learning. In *DEEM@SIGMOD 2018*, pages 31–34.

[133] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS 2016*, pages 3567–3575.

[134] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(43):1297–1322, 2010.

[135] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11), 2017.

[136] E. K. Rezig, L. Cao, G. Simonini, M. Schoemans, S. Madden, N. Tang, M. Ouzzani, and M. Stonebraker. Dagger: a data (not code) debugger. In *CIDR 2020*.

[137] F. Rodrigues and F. C. Pereira. Deep learning from crowds. In *AAAI 2018*, pages 1611–1618.

[138] Y. Roh, G. Heo, and S. E. Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[139] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *(ICML 2001*, pages 441–448.

[140] C. D. Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang. Deepdive: Declarative knowledge base construction. *SIGMOD Rec.*, 45(1):60–67, 2016.

[141] C. D. Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang. Incremental knowledge base construction using deepdive. *VLDB J.*, 26(1):81–105, 2017.

[142] A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and A. Y. Halevy. Crowd-powered find algorithms. In *ICDE 2014*, pages 964–975.

[143] R. E. Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406. Citeseer, 1999.

[144] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD Conference 2016*, pages 3–18.

[145] M. Schleich, D. Olteanu, M. A. Khamis, H. Q. Ngo, and X. Nguyen. A layered aggregate engine for analytics workloads. In *SIGMOD Conference 2019*, pages 1642–1659.

[146] T. Sellam, K. Lin, I. Huang, M. Yang, C. Vondrick, and E. Wu. Deepbase: Deep inspection of neural networks. In *SIGMOD 2019*, pages 1117–1134, 2019.

[147] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[148] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. In *EMNLP 2008*, pages 1070–1079.

[149] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In *NIPS 2007*, pages 1289–1296.

[150] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT 1992*, pages 287–294.

[151] V. Shah, A. Kumar, and X. Zhu. Are key-foreign key joins safe to avoid when learning high-capacity classifiers? *Proc. VLDB Endow.*, 11(3):366–379, 2017.
[152] A. Shanbhag, S. Madden, and X. Yu. A study of the fundamental performance characteristics of gpus and cpus for database analytics. In *SIGMOD Conference 2020*, pages 1617–1632.
[153] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.
[154] R. Souza, L. G. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. V. Brazil, M. Moreno, P. Valduriez, et al. Workflow provenance in the lifecycle of scientific machine learning. *arXiv preprint arXiv:2010.00330*, 2020.
[155] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning. In *SoCC 2015*, pages 368–380.
[156] D. Steinberg and P. Colla. Cart: classification and regression trees. *The top ten algorithms in data mining*, 9:179, 2009.
[157] R. Tate, M. Stepp, Z. Tatlock, and S. Lerner. Equality saturation: a new approach to optimization. In *POPL 2009*, pages 264–276, New York, NY, USA. ACM.
[158] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011.
[159] P. Varma, D. Iter, C. De Sa, and C. Ré. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, HILDA'17, New York, NY, USA, 2017. Association for Computing Machinery.
[160] M. Vartak, J. M. F. da Trindade, S. Madden, and M. Zaharia. Mistique: A system to store and query model intermediates for model diagnosis. SIGMOD '18, page 12851300, 2018.
[161] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. HILDA '16, 2016.
[162] V. Verroios, P. Lofgren, and H. Garcia-Molina. tdp: An optimal-latency budget allocation strategy for crowdsourced MAXIMUM operations. In *SIGMOD 2015*, pages 1047–1062.
[163] J. Wang, C. Chai, J. Liu, and G. Li. FACE: A normalizing flow based cardinality estimator. *Proc. VLDB Endow.*, 15(1):72–84, 2021.
[164] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, 2012.
[165] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD 2014*, pages 469–480.
[166] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD 2013*.
[167] K. Wang, K. Zhang, Y. Yuan, S. Ma, R. Lee, X. Ding, and X. Zhang. Concurrent analytical query processing with gpus. *Proc. VLDB Endow.*, 7(11):1011–1022, 2014.
[168] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska. Superneurons: dynamic GPU memory management for training deep neural networks. In A. Krall and T. R. Gross, editors, *SIGPLAN 2018*.
[169] P. Wang, R. Shea, J. Wang, and E. Wu. Progressive deep web crawling through keyword queries for data enrichment. In *SIGMOD 2019*.
[170] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1231–1245, 2015.
[171] Y. R. Wang, S. Hutchison, D. Suciu, B. Howe, and J. Leang. SPORES: sum-product optimization via relational equality saturation for large scale linear algebra. *Proc. VLDB Endow.*, 13(11):1919–1932, 2020.
[172] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. R. Movellan. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS 2009*, pages 2035–2043.
[173] W. Wu, L. Flokas, E. Wu, and J. Wang. Complaint-driven training data debugging for query 2.0. In *SIGMOD Conference 2020*, 2020.
[174] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. G. Parameswaran. Helix: Accelerating human-in-the-loop machine learning. *Proc. VLDB Endow.*, 11(12):1958–1961, 2018.
[175] D. Xin, S. Macke, L. Ma, J. Liu, S. Song, and A. G. Parameswaran. Helix: Holistic optimization for accelerating iterative machine learning. *Proc. VLDB Endow.*, 12(4):446–460, 2018.
[176] Z. Xu, R. Akella, and Y. Zhang. Incorporating diversity and density in active learning for relevance feedback. In *ECIR 2007*, volume 4425 of *Lecture Notes in Computer Science*, pages 246–257.
[177] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD 2012*, pages 97–108.
[178] J. Yang, J. Fan, Z. Wei, G. Li, T. Liu, and X. Du. A game-based framework for crowdsourced data labeling. *VLDB J.*, 29(6), 2020.
[179] K. Yang, Y. Gao, L. Liang, B. Yao, S. Wen, and G. Chen. Towards factorized SVM with gaussian kernels over normalized data. In *ICDE 2020*. IEEE.
[180] Y. Yang, M. P. Phothilimtha, Y. R. Wang, M. Willsey, S. Roy, and J. Pienaar. Equality saturation for tensor graph superoptimization. *CoRR*, abs/2101.01332, 2021.
[181] X. Yu, G. Li, C. Chai, and N. Tang. Reinforcement learning with tree-lstm for join order selection. In *ICDE 2020*, pages 1297–1308.
[182] Y. Yuan, R. Lee, and X. Zhang. The yin and yang of processing data warehousing queries on GPU devices. *Proc. VLDB Endow.*, 6(10), 2013.
[183] C. Zhan, M. Su, C. Wei, X. Peng, L. Lin, S. Wang, Z. Chen, F. Li, Y. Pan, F. Zheng, and C. Chai. Analyticdb: Real-time OLAP database system at alibaba cloud. *Proc. VLDB Endow.*, 12(12):2059–2070, 2019.
[184] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)*, 41(1):1–32, 2016.
[185] H. Zhang, C. Chai, A. Doan, P. Koutris, and E. Arcaute. Manually detecting errors for data cleaning using adaptive crowdsourcing strategies. In A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang, editors, *EDBT 2020*, pages 311–322.
[186] M. Zhang and K. Chakrabarti. Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD 2013*, pages 145–156.
[187] Y. Zhang and Z. G. Ives. Finding related tables in data lakes for interactive data science. In *SIGMOD Conference 2020*.
[188] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13001–13008, 2020.
[189] D. Zhou, Q. Liu, J. C. Platt, C. Meek, and N. B. Shah. Regularized minimax conditional entropy for crowdsourcing. *arXiv preprint arXiv:1503.07240*, 2015.
[190] X. Zhou, C. Chai, G. Li, and J. SUN. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.
[191] X. Zhou, G. Li, C. Chai, and J. Feng. A learned query rewrite system using monte carlo tree search. *Proc. VLDB Endow.*, 15(1):46–58, 2021.

Chengliang Chai is currently a Postdoc researcher in Computer Science and Technology of Tsinghua University. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2020. His research interests mainly include data cleaning and integration, crowdsourcing and database system.

Jiayi Wang received his bachelor's degree in Software Engineering from Beihang University in 2020. He is currently a master student in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include machine learning for database and query processing.

Yuyu Luo received his bachelor's degree in Software Engineering from the University of Electronic Science and Technology of China in 2018. He is currently a PhD student in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include database system and data visualization.

Zeping Niu received his bachelors degree in Computer Science from Tsinghua University in 2020. He is currently a PhD student in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include data management for AI.

Guoliang Li is currently working as a professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include database system, data cleaning and integration, spatial databases and crowdsourcing.