Rock: Cleaning Data by Embedding ML in Logic Rules

Xianchun Bao[*]
Zian Bao
baoxianchun@sics.ac.cn
baozian@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Binbin Bie
QingSong Duan
biebinbin@sics.ac.cn
duanqingsong@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Wenfei Fan
Shenzhen Institute of Computing Sciences, China
University of Edinburgh
United Kingdom
Beihang University, China
wenfei@inf.ed.ac.uk

Hui Lei
Daji Li
leihui@sics.ac.cn
lidaji@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Wei Lin
Peng Liu
linwei@sics.ac.cn
liupeng@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Zhicong Lv
Mingliang Ouyang
lvzhicong@sics.ac.cn
ouyangmingliang@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Shuai Tang
Yaoshu Wang[†]
tangshuai@sics.ac.cn
yaoshuw@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Qiyuan Wei
Min Xie
werty@sics.ac.cn
xiemin@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Jing Zhang
zhangjing@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Xin Zhang
zhangxinzx@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Runxiao Zhao
zhaorunxiao@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

Shuping Zhou
zhoushuping@sics.ac.cn
Grandhoo Inc., China
Shenzhen Institute of Computing Sciences, China

ABSTRACT

We introduce Rock, a system for cleaning relational data. Rock implements a framework that unifies machine learning (ML) and logic deduction by embedding ML classifiers in rules as predicates. In a unified process, it identifies tuples that refer to the same real-world entity, catches semantic inconsistencies among the entities, deduces the timeliness of the attribute values of the entities, and imputes missing values by possibly extracting data from knowledge graphs. That is, Rock conducts entity resolution, conflict resolution, incomplete information imputation and timeliness deduction in the same process, makes use of their interactions and improves the overall quality of the data. Moreover, Rock supports methods, batch and incremental, for discovering rules from real-life data, detecting errors with the learned rules, accumulating ground truth, and fixing the errors, such that the corrections are logical consequences of the rules and ground truth. We present the design and implementation of Rock. We evaluate the scalability and accuracy of Rock, and share lessons learned from a variety of real-life applications.

[*]Author names are listed in alphabetical order.
[†]Corresponding author.

CCS CONCEPTS

• **Information systems → Information integration**.

KEYWORDS

Data quality; entity resolution; conflict resolution; missing value imputation; timeliness deduction; error detection; error correction

ACM Reference Format:
Xianchun Bao, Zian Bao, Binbin Bie, QingSong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, Mingliang Ouyang, Shuai Tang, Yaoshu Wang, Qiyuan Wei, Min Xie, Jing Zhang, Xin Zhang, Runxiao Zhao, and Shuping Zhou. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3626246.3653372

1 INTRODUCTION

Real-life data is often dirty, evidenced by duplicates, conflicts, incomplete information and obsolete values commonly found in our datasets. Dirty data is costly. Gartner assessed that poor data quality is responsible for an average of $15 million per year in losses for organizations [46]; IBM estimated that dirty data cost the US $3.1 trillion in 2016 alone (cf. [75]); and inaccurate customer data costs organizations 6% of their annual revenues [78]. Dirty data has been a longstanding challenge, and remains a clear and present danger to big data analytics. Indeed, data-driven decisions based on dirty data can be worse than making decisions with no data.

Challenges. To cope with dirty data, there has been a large body of work [10–13, 15, 22, 23, 25, 30, 32, 33, 47, 48, 57, 60, 66, 68, 73, 76, 82, 87, 88, 93]. Several data cleaning systems have also been developed, from early tools for imputing census data [43, 44] to recent systems such as Talend [5], Amazon Glue [2] and Informatica [4].

However, to develop a system that is effective in cleaning real-life dirty data, several immediate issues have to be addressed.

Challenge 1: Machine learning or logic deduction? Existing research and systems on data quality are typically approached via either logic rules [12, 13, 15, 22, 30, 32, 33, 47, 57, 88] or machine learning (ML) [10, 11, 23, 25, 48, 60, 66, 68, 73, 76, 82, 87, 93]. Unfortunately, none of the two is super to the other. On the one hand, it is hard to find a small number of logic rules that cover different cases and data in practice. On the other hand, ML solutions are probabilistic and hard to interpret; practitioners may not want to deploy ML models when cleaning critical data such as medical records.

Is it possible to unify ML and logic rules in the same process, to benefit from both? How effective can such a uniform framework be, compared to logic deduction and ML predictions alone?

Challenge 2: Functionality. Previous data cleaning systems have mostly focused on two primitive issues of data quality:

- *entity resolution* (ER): to determine whether two tuples refer to the same real-world entity, in order to catch duplicates; and
- *conflict resolution* (CR): to catch semantic inconsistencies among attribute values of the entities, and resolve the conflicts.

However, there are two other critical issues of data quality:

- *missing value imputation* (MI): to enrich tuples in our datasets by filling in the missing values (null); and
- *timeliness deduction* (TD): to deduce temporal orders on attribute values, and infer the latest attribute values of each entity.

The need for studying these is evident, *e.g.,* missing data is common in epidemiological research, where 42.5% of records are incomplete [51], and "58% of organizations make decisions based on outdated data" [28], although "as a healthcare, retail, or financial business you cannot afford to make decisions based on yesterday's data".

Moreover, these critical issues interact with each other. On the one hand, deducing missing values and temporal orders help us identify entities and fix inconsistencies. On the other hand, ER and CR facilitate it to enrich tuples by instantiating missing values and deduce timeliness by providing more correlated values.

Is it possible to support all of ER, CR, MI and TD in a system? Can we exploit their interactions to improve the overall data quality?

Challenge 3: Performance. To clean dirty data, a data quality system should support the following at the very least:

- *rule discovery*: to discover logic rules and/or train ML models with (possibly dirty and large) real-life data;
- *error detection*: to catch errors (duplicates, inconsistencies, missing and stale values) with the rules/models learned; and
- *error correction*: to fix the errors detected (merge duplicates, resolve conflicts, fill in null values and deduce the latest values).

Criteria for an effective data quality system include (a) scalability, *i.e.,* the capability to scale with large-scale datasets, and (b) accuracy, *i.e.,* guarantees to minimize false positives and false negatives for error detection and correction. These are highly nontrivial. Discovery of even functional dependencies (FDs) easily takes exponential time and yields an excessive number of FDs [52], and error correction with accuracy bounds is intractable even with FDs [17].

Is it possible to have a system that supports ER, CR, MI and TD, employs both ML models and logic rules, and at the same time, is able to scale with large datasets? Can it guarantee that the fixes are "certain"? That is, it ensures that each update to the data corrects (fixes) an existing error and does not introduce new errors. This is a must when we clean critical data such as medical records.

Rock: A system for data cleaning. In response to the practical need, we have developed Rock [6], a system for cleaning relational data. Rock has been deployed at banks, logistics, mobile operators and e-commerce, among other places; it has proven effective in a variety of applications. Rock has the following unique properties.

(1) A uniform framework. Rock implements a uniform framework that unifies ML and logic deduction. It proposes REE^{++}s, an extension of Entity Enhancing Rules (REEs) [36, 41] by incorporating temporal orders, correlation models and heterogeneous entity resolution (HER) [31] as predicates. REE^{++}s subsume conditional functional dependencies (CFDs) [32], denial constraints (DCs) [13] and matching dependencies (MDs) [30] as special cases; as opposed to previous data quality rules, REE^{++}s may embed ML classifiers as predicates. In this way, Rock extends traditional logic deduction with ML models, and benefit from logic interpretation of the rules.

(2) A unified process. In addition to ER and CR, Rock supports missing data imputation (MI) and timeliness deduction (TD), by integrating logic deduction, temporal ranking, ML correlation models and data extraction from knowledge graphs. All the four tasks can be expressed as REE^{++}s, and hence ER, CR, MI and TD can be conducted in the same process, and interact with each other. In particular, by supporting HER [31] for aligning entities across relations and graphs, Rock can extract properties from graphs to enrich relations.

(3) Scalability. Rock implements algorithms for rule discovery, error detection and error correction [36, 37, 41]. It supports a *batch* mode to conduct these on static datasets, and an *incremental* mode in response to updates. All the algorithms are *parallelly scalable* [62], *i.e.,* they guarantee to reduce runtime when more machines are used; in principle, Rock is able to scale when the data grows big. Rock also supports other techniques to deal with big data, *e.g.,* top-k rule discovery [37] and sampling with accuracy guarantees [36].

(4) Certain fixes. By embedding ML models in logic rules, Rock offers high accuracy in error detection and error correction. Moreover, it accumulates ground truth, *i.e.,* validated data, when correcting errors; it extends the chase [79] with REE^{++}s and conflict resolution, and references the ground truth when fixing errors. As a consequences, it guarantees that fixes generated are logical consequences of the rules and ground truth. The fixes are guaranteed to be correct, known as *certain fixes* [38, 40], as long as the rules and ground truth are correct, and if embedded ML predictions are accurate.

Organization. We present Rock as follows:

- entity enhancing rules REE^{++}s underlying Rock (Section 2);
- the system architecture of Rock (Section 3);
- the process that unifies ER, CR, MI and TD (Section 4);
- the implementation and optimization of Rock (Section 5); and
- real-life applications and evaluation of Rock (Section 6).

We will discuss related data quality systems in Section 7, and present our plan for extending Rock in Section 8.

2 EXTENDED ENTITY ENHANCING RULES

This section presents REE^{++}s, extended Entity Enhancing Rules. We first review REEs studied in [36, 37, 39, 41] for ER and CR (Section 2.1). We then extend REEs for deducing timeliness (TD, Section 2.2) and imputing missing values (MI, Section 2.3). This section reveals the entire class of REE^{++}s underlying Rock for the first time.

Preliminaries. We define REE^{++}s on a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where R_j is a relation schema $R(A_1 : \tau_1, \ldots, A_k : \tau_k)$, and each A_i is an attribute of type τ_i. An instance $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_i is a relation of R_i. Following [21], we assume that each tuple t has an EID attribute, identifying the entity that t represents.

We represent a knowledge graph as $G = (V, E, L)$, where (a) V is a finite set of vertices, (b) $E \subseteq V \times V$ is a set of edges, and (c) L is a function such that for each vertex $v \in V$ (resp. edge $e \in E$), $L(v)$ (resp. $L(e)$) is a vertex (resp. edge) label. Here an edge label typifies predicates while vertex labels may carry values.

A *label path* is a list $\rho = (l_1, \ldots, l_n)$ of edge labels. A *match* of ρ in G is a list $(v_0, v_1, \ldots, v_n)$ such that (v_{i-1}, l_{i-1}, v_i) is an edge in G.

2.1 REEs for ER and CR

The REEs reported in [36, 39, 41] have the following predicates.

Predicates. *Predicates* over schema $\mathcal{R}$ are defined as follows:

$$p ::= R(t) \mid t.A \oplus c \mid t.A \oplus s.B \mid \mathcal{M}(t[\bar{A}], s[\bar{B}]),$$

where $\oplus$ is an operator in $\{=, \neq, <, \leq, >, \geq\}$. Following tuple relational calculus [8], (a) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is a *tuple variable bounded by* $R(t)$; (b) $t.A$ denotes an attribute of t when t is bounded by $R(t)$ and A is an attribute in R; (c) $t.A \oplus c$ is *a constant predicate* when c is a value in the domain of A; and (d) $t.A \oplus s.B$ compares *compatible* attributes $t.A$ and $s.B$, *i.e.,* tuple t (resp. s) is bounded by $R(t)$ (resp. $R'(s)$), and $A \in R$ and $B \in R'$ have the same type. Moreover, (e) $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ is *an ML predicate*, where $t[\bar{A}]$ and $s[\bar{B}]$ are vectors of pairwise compatible attributes.

ML models. Here $\mathcal{M}$ can be any existing ML model that returns a Boolean value, *e.g.,* $\mathcal{M}_{\text{reg}} \geq \delta$ for the strength of a regression model $\mathcal{M}_{\text{reg}}$ and a predefined threshold δ. We consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [24], for text classification; (2) ER models and link prediction models, *e.g.,* Bert [24] for semantic matching; and (3) models for error detection and correction, *e.g.,* generative models [92], holistic ML models [66, 76] and statistical models [90, 91].

REEs. An *entity enhancing rule* (REE) φ over $\mathcal{R}$ is defined as

$$\varphi : X \to p_0,$$

where X is a conjunction of *predicates* over $\mathcal{R}$, and p_0 is a predicate over $\mathcal{R}$ such that all tuple variables in φ are bounded in X. We refer to X as the *precondition* of φ, and p_0 as the *consequence* of φ.

Example 1: Consider an e-commerce database with self-explained schemas Person(pid, last_name (LN), first_name (FN), gender, home, status, spouse), Store(sid, name, type, location, accu_sales (accumulated sales), area_code) and Transaction(pid, sid, commodity(com), manufactory(mfg), price, date) in Tables 1-3. The erroneous values are in bold. Three simple REEs are given as follows.

φ_1 : $\text{Trans}(t) \wedge \text{Trans}(s) \wedge \mathcal{M}_{\text{ER}}(t.\text{com}, s.\text{com}) \wedge t.\text{date} = s.\text{date} \wedge t.\text{sid} = s.\text{sid} \to t.\text{pid} = s.\text{pid}$, where $\mathcal{M}_{\text{ER}}$ is an ER model to identify two commodities that use the same discount code. The rule identifies two persons since the same discount code can be used at most once in the same store by a person during the discount period.

φ_2 : $\text{Trans}(t) \wedge \text{Trans}(s) \wedge t.\text{com} = s.\text{com} \to t.\text{mfg} = s.\text{mfg}$. This REE^{++}s says that the manufactory must be the same if the commodities are the same; it can fix the erroneous manufactory.

φ_3 : $\text{Person}(t) \wedge \text{Person}(s) \wedge X \to \mathcal{M}_{\text{ad}}(t[\text{home}], s[\text{home}])$, where $\mathcal{M}_{\text{ad}}$ is a model to check address closeness, $X = \bigwedge_{A \in \mathcal{T}} t.A = s.A$ and $\mathcal{T}$ is a set of attributes about home addresses (not shown), including zipcode and neighborhood information. Here conditions in X explain why $\mathcal{M}_{\text{ad}}(t[\text{home}], s[\text{home}])$ predicts true. □

Semantics. Consider an instance $\mathcal{D}$ of $\mathcal{R}$. A *valuation* h of tuple variables of φ in $\mathcal{D}$, or simply a valuation of φ, is a mapping that instantiates t in each $R(t)$ with a tuple in a relation D of $\mathcal{D}$.

We write $h \models p$ for predicate p: (1) if p is $R(t)$, $t \oplus c$ or $t.A \oplus s.B$, then $h \models p$ is interpreted as in tuple relational calculus following the standard semantics of first-order logic [8]; (2) if p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, then $h \models p$ if $\mathcal{M}$ predicts true on $(h(t)[\bar{A}], h(s)[\bar{B}])$.

Given a conjunction X of predicates, we say $h \models X$ if for *all* predicates p in X, $h \models p$. Given an REE φ, we write $h \models \varphi$ such that if $h \models X$, then $h \models p_0$. An instance $\mathcal{D}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{D} \models \varphi$, if *for all* valuations h of tuple variables of φ in $\mathcal{D}$, $h \models \varphi$. We write $\mathcal{D} \models \Sigma$ for a set Σ of REEs if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Properties. (1) As shown in [39], REEs subsume CFDs, DCs and MDs as special cases. (2) As indicated in Example 1, REEs may unify ER, CR and association analysis. An REE may carry multiple tuple variables for collective analysis across tables [16]. (3) As shown by φ_3, for certain ML models $\mathcal{M}$, one can discover logic conditions X to provide high-level rational behind predictions of $\mathcal{M}$ in an REE of the form $X \to \mathcal{M}(t[\bar{A}], s[\bar{B}])$, when $\mathcal{M}$ is the rule consequence. (4) We can embed $\mathcal{M}$ as predicates in precondition X (see φ_1), not to feed REEs as features to ML models. As shown in [29], this strategy can filter false positives/negatives of predictions of $\mathcal{M}$ with $X = \mathcal{M}(t[\bar{A}], s[\bar{B}]) \wedge X_1$, *i.e.,* we can enrich $\mathcal{M}$ with extra conditions X_1.

2.2 REE^{++}s for Deducing Timeliness

In practice we often need to determine temporal orders on attribute values. To formalize this intuition, we start with some notations.

Temporal relations. A *temporal relation* is (D, T), where (a) D is a normal relation of schema R, and (b) T is a partial function that associates a timestamp $T(t[A])$ with the A-attribute of a tuple t in D.

The timestamp indicates that at the time $T(t[A])$, the A-attribute value of tuple t is correct and up-to-date. A temporal relation extends a relation with available timestamps. In the same tuple t, $t[A]$ and $t[B]$ may bear different timestamps for different A and B, since different attributes of t may come from different data sources.

Temporal orders. A *temporal order* on attribute A of D is a partial order $\preceq_A$ such that for all tuples t_1 and t_2 in D, $t_2 \preceq_A t_1$ if the value in $t_1[A]$ is at least as current as $t_2[A]$. Note that $t_2 \preceq_A t_1$ ranks the timeliness of the A-attributes of tuples t_1 and t_2, not values detached from the tuples. Similarly, a strict partial order $t_2 \prec_A t_1$ says that $t_1[A]$ is more current than $t_2[A]$. We represent $\preceq_A$ as a set of tuple pairs such that $(t_2, t_1) \in \preceq_A$ iff $t_2 \preceq_A t_1$; similarly for $\prec_A$.

In particular, if $T(t_1[A])$ and $T(t_2[A])$ are both defined and if $T(t_2[A]) \leq T(t_1[A])$, then $t_2 \preceq_A t_1$, *i.e.,* $t_1[A]$ is confirmed at a later timestamp and is thus considered at least as current as $t_2[A]$.

tid	pid	LN	FN	gender	home	status	spouse
t_1	p_1	Jones	Christine	F	5 Beijing West Road	single	n/a
t_2	p_2	Smith	Christine	F	**5 West Road**	single	p_3
t_3	p_2	Smith	Christine	F	12 Beijing Road	married	p_4
t_4	p_3	Smith	George	M	12 Beijing Road	married	p_2
t_5	p_4	Smith	George	M	null	null	null

Table 1: Example Person **relation** D_1

tid	sid	name	type	location	accu_scales	area_code
t_6	s_1	Apple Jingdong Self-run	Electron.	Beijing	15M	null
t_7	s_2	Apple Taobao Flagship	Electron.	null	null	null
t_8	s_3	Huawei Flagship	Electron.	Beijing	11M	null
t_9	s_4	Huawei	**Sports**	Shanghai	10M	021
t_{10}	s_5	Nike China	Sports	Shanghai	null	021

Table 2: Example Store **relation** D_2

tid	pid	sid	com	mfg	price	date
t_{11}	p_1	s_2	IPhone 13	Apple	9000	2020-12-18
t_{12}	p_1	s_1	IPhone 14 (Discount ID 41)	Apple	6500	2021-11-11
t_{13}	p_2	s_1	IPhone 14 (Discount Code 41)	Apple	null	2021-11-11
t_{14}	p_3	s_3	Mate X2 (Limited Sold)	Huawei	5200	2023-8-12
t_{15}	p_4	s_4	Mate X2 (Limited Sold)	**Apple**	null	2023-8-12

Table 3: Example Transaction (Trans**) relation** D_3

Temporal instances. Consider an instance $\mathcal{D} = (D_1, \ldots, D_m)$ of database schema $\mathcal{R}$, and temporal relations (D_i, T_i) for $i \in [1, m]$. A *temporal instance* $\mathcal{D}_t$ *of schema* $\mathcal{R}$ is $(\mathcal{D}, \preceq_{A_1}, \ldots, \preceq_{A_n}, T)$, where T is $\bigcup_{i\in[1,m]} T_i$, A_i ranges over attributes of $\mathcal{R}$ ($i \in [1, n]$), and each $\preceq_{A_i}$ is a temporal order on A_i. We assume *w.l.o.g.* that attributes are distinct across relations, *e.g.,* prefixed by its relation name.

That is, the temporal instance $\mathcal{D}_t$ extends $\mathcal{D}$ with explicit partial temporal orders $\preceq_{A_i}$, one for each attribute A_i in $\mathcal{R}$ ($i \in [1, n]$).

ML ranking model. Rock has trained a pairwise ranking model [42], referred to as $\mathcal{M}_{\text{rank}}$, by interleaving model learning and verification with currency constraints [34]. Given any tuples t_1 and t_2 of a relation D and any attribute A of D, $\mathcal{M}_{\text{rank}}(t_1, t_2, \otimes_A)$ returns true if it predicts $t_1 \otimes_A t_2$, and false otherwise, where $\otimes_A$ is either $\preceq_A$ or $\prec_A$. We find that $\mathcal{M}_{\text{rank}}$ has F-measure consistently above 0.80.

Extending REEs. REE^{++}s also support temporal predicates below:

$$p ::= t \preceq_A s \;\mid\; t \prec_A s,$$

for tuple variables t and s bounded by the same relation schema R that has attribute A, in addition to the predicates listed in Section 2.1.

An REE^{++} also has the form $X \rightarrow p_0$. The semantics of such REE^{++}s is a straightforward extension of the one in Section 2.1.

In particular, REE^{++}s allow $\mathcal{M}_{\text{rank}}(t_1, t_2, \otimes_A)$ as an ML predicate, where $\mathcal{M}_{\text{rank}}$ is the ML temporal ranking model of [42], $\otimes_A$ is $\preceq_A$ or $\prec_A$, and t_1 (resp. t_2) is bounded by $R(t_1)$ (resp. $R(t_2)$).

Example 2: REE^{++}s with temporal predicates can express interesting properties, *e.g.,* monotonicity, comonotonicity and correlation.

φ_4 : $\text{Person}(t) \wedge \text{Person}(s) \wedge t.\text{status} = \text{“single”} \wedge s.\text{status} = \text{“married”} \rightarrow t \preceq_{\text{status}} s$. It says marital status only changes *monotonically, i.e.,* from single to married, not the other way around [18].

φ_5 : $\text{Person}(t) \wedge \text{Person}(s) \wedge t \preceq_{\text{status}} s \rightarrow t \preceq_{\text{home}} s$, *i.e.,* $\preceq_{\text{status}}$ and $\preceq_{\text{home}}$ are often comonotonic: when the marital status of a person changes, this person may move to a different house.

φ_6 : $\text{Store}(t) \wedge \text{Store}(s) \wedge t.\text{location} = \text{“Shanghai”} \wedge s.\text{location} = \text{“Beijing”} \wedge t.\text{accu\_sales} \leq s.\text{accu\_sales} \rightarrow t \preceq_{\text{location}} s$. Here φ_6 correlates multiple attributes to capture implicit ordering. Note that a store can move from Beijing to Shanghai and further from Shanghai to Beijing. We can use its accumulated sales as an additional hint, which changes monotonically, to deduce its current location. □

2.3 REE^{++}s for Imputing Missing Data

Rock fills in missing values by unifying three strategies: logic, ML prediction and data extraction from knowledge graphs. It conducts these by using REE^{++}s with the following additional predicates.

Predicates. In addition to predicates given earlier, REE^{++}s support:

$$p \;::=\; \text{vertex}(x, G) \mid \text{HER}(t, x) \mid \text{match}(t.A, x.\rho) \mid t[A] = \text{val}(x.\rho) \mid \mathcal{M}_c(t[\bar{A}], t[B]) \;\geq\; \delta \mid \mathcal{M}_c(t[\bar{A}], t[B]{=}c) \;\geq\; \delta \mid t[B] = \mathcal{M}_d(t[\bar{A}], B).$$

Here (a) x in $\text{vertex}(x, G)$ is a variable denoting a vertex in knowledge graph G, referred to as a *variable bounded by* $\text{vertex}(x, G)$. (b) If x is bounded by $\text{vertex}(x, G)$ and t is bounded by $R(t)$, $\text{HER}(t, x)$ is a Boolean function that returns true if tuple t and vertex x refer to the same entity. (c) If ρ is a label path and if x and t are bounded as above, $\text{match}(t.A, x.\rho)$ checks whether the path ρ from vertex x encodes the A-attribute of tuple t. (d) If t and x are bounded as above and $\text{match}(t.A, x.\rho)$ returns true, $t[A] = \text{val}(x.\rho)$ indicates that the A-attribute of t takes the value (label) of the last vertex v on the match of ρ from vertex x. (e) ML model $\mathcal{M}_c$ assesses the strength of the correlation between (partial) tuple $t[\bar{A}]$ and the B-attribute value $t[B]$; in $\mathcal{M}_c(t[\bar{A}], c) \geq \delta$, δ is a strength threshold. (f) Given a partial tuple $t[\bar{A}]$, ML model $\mathcal{M}_d$ predicts a value for its B-attribute.

We remark the following about these new predicates.

(1) Several methods for implementing $\text{HER}(t, x)$ are already in place, *e.g.,* rule-based JedAI [70], parametric simulation [31], and ML models Silk [54] and MAGNN [45]. Rock supports [31] as a Boolean function to check whether a tuple in a relation and a vertex in a graph refer to the same entity (heterogeneous entity resolution).

(2) Rock has implemented $\text{match}(t.A, x.\rho)$ by using a Long-Short Term Memory (LSTM) network [50] as shown in [31].

(3) Predicates $\text{vertex}(x, G)$, $\text{HER}(t, x)$, $\text{match}(t.A, x.\rho)$ and $t[A] = \text{val}(x.\rho)$ aim to identify entities across relation D and knowledge graph G, and extract data from G to instantiate the missing values of attribute $t[A]$ in D. We refer to them as *extraction predicates.*

(4) Predicate $\mathcal{M}_c(t[\bar{A}], t[B]) \geq \delta$ and $\mathcal{M}_c(t[\bar{A}], c) \geq \delta$ assess correlation between values. Here $t[B] = \mathcal{M}_d(t[\bar{A}], B)$ suggests a value for (missing) attribute B. We refer to them as *correlation predicates.*

Extended rules. An REE^{++} φ also has the form $X \rightarrow p_0$ such that all tuple variables and vertex variables in φ are bounded in X. Intuitively, when p_0 is $t[A] = \text{val}(x.\rho)$ or $t[B] = \mathcal{M}_d(t[\bar{A}], B)$, the rule fills in missing values by extracting data from a knowledge base or employing ML prediction of an accurate model $\mathcal{M}_d$, respectively.

Example 3: The following REE^{++}s impute missing values.

φ_7 : $\text{Store}(t) \wedge \text{vertex}(x, \text{Wiki}) \wedge \text{HER}(t, x) \wedge \text{match}(t[\text{location}], x.(\text{LocationAt})) \rightarrow t[\text{location}] = \text{val}(x.(\text{LocationAt}))$. It says that if a store t matches a vertex x in WikiPedia and if x reaches vertex v via path $\rho = (\text{LocationAt})$, let $t[\text{location}]$ take $L(v)$ as its value.

φ_8 : $\text{Trans}(t) \wedge \text{null}(t[\text{price}]) \rightarrow t[\text{price}] = \mathcal{M}_d(t[\bar{A}], \text{price})$, where $\text{null}(t[\text{price}])$ is a syntactic abbreviation to check whether $t[\text{price}]$ carries null value and $\bar{A}$ is the set of all validated values of t. This REE^{++}s predicts the missing price of t via validated values. □

Semantics. We extend the notion of valuation to be a mapping h that instantiates each tuple variable t with a tuple in a database $\mathcal{D}$, and each vertex variable x with a vertex in a knowledge graph G.

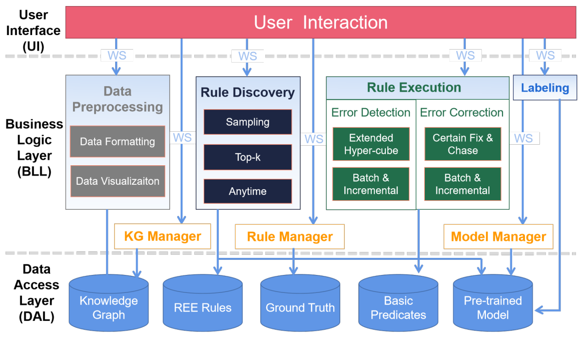


Figure 1: Architecture of Rock

For the additional predicates p, a valuation h satisfies p, denoted by $h \models p$, if the following is satisfied. (a) If p is $\mathsf{HER}(t, x)$, then $h(t)$ and $h(x)$ refer to the same entity as determined by the Boolean function HER. (b) If p is $\mathsf{match}(t.A, x.\rho)$, then the labels on the path ρ match the attribute A of schema R, and there exists a match of path ρ from $h(x)$, where t is bounded by $R(t)$. (c) If p is $t[A] = \mathsf{val}(x.\rho)$, then the match of ρ from $h(x)$ reaches a vertex v in G, and the value of $h(t)[A]$ is equal to the value (label) of v. (d) If p is $\mathcal{M}_c(t[\bar{A}], c) \geq \delta$ (resp. $\mathcal{M}_c(t[\bar{A}], t[B]) \geq \delta$), let d be the strength of the correlation between $h(t)[\bar{A}]$ and c (resp. $t[B]$) assessed by $\mathcal{M}_c$, then $d \geq \delta$. (e) If p is $t[B] = \mathcal{M}_d(t[\bar{A}], B)$, then the value of $t[B]$ is equal to the B-attribute value suggested by $\mathcal{M}_d$ for partial tuple $t[\bar{A}]$.

3 SYSTEM ARCHITECTURE

This section presents the architecture and workflow of Rock.

Architecture. As shown in Figure 1, Rock is developed based on a three-tier architecture. User interface (UI) is the topmost level. It displays standard graphical interfaces, receives user requests, communicates with other layers via Web socket (WS) and returns the results to users. The business logic layer (BLL) conducts processing. It also moves and processes data between the two surrounding layers. The data access layer (DAL) provides APIs for BLL to access and manage the stored data. The retrieved data is passed back to BLL for processing, and is eventually returned back to the users.

Workflow. Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$, Rock first discovers a set Σ of REE^{++}s over $\mathcal{R}$ *offline*. It then detects and fixes errors in $\mathcal{D}$ online, by employing the REE^{++}s in Σ. If the users request to correct the errors, Rock chases $\mathcal{D}$ with Σ to conduct "deep" cleaning in parallel. Moreover, the users may opt to employ Rock to monitor changes to $\mathcal{D}$, and incrementally detect and fix errors in response to updates. These are carried out by the following key modules.

Rule discovery. Rock supports three algorithms for mining/learning REE^{++}s from a (possibly large) dataset $\mathcal{D}$. Rule mining methods often return excessive candidates and incur prohibitive cost. To overcome these, Rock supports (a) a top-k method [37] that learns a model for ranking REE^{++}s based on both objective measures (confidence, support) and subjective measures (users' preference and unexpectedness), and mines top-ranked REE^{++}s; (b) an anytime algorithm [37] for successive REE^{++} mining via lazy evaluation, and (c) a multi-round sampling method [36] that mines REE^{++}s from a fraction of $\mathcal{D}$ and guarantees their accuracy (precision and recall) under a probabilistic bound. The algorithms substantially reduce the discovery cost and find REE^{++}s that meet the need of different users.

Error detection. Given a set Σ of REE^{++}s and a dataset $\mathcal{D}$, Rock detects errors in $\mathcal{D}$ as violations of REE^{++}s in Σ (see Section 4.2). The errors includes duplicates, semantic inconsistencies, obsolete values and missing values. Rock also incrementally detects errors in response to updates $\Delta\mathcal{D}$ to $\mathcal{D}$. It implements parallel algorithms and parallel incremental algorithms for error detection, which extend the algorithms of [41], both parallelly scalable to scale with large $\mathcal{D}$.

Error correction. To correct the errors detected, Rock extends the classic chase [79] with conflict resolution strategies [35] to conduct "deep cleaning", and recursively propagate the corrections. It conducts ER, CR, TD and MI in the same chase process. Rock accumulates ground truth in the process and references the ground truth when fixing errors. The chase is *Church-Rosser* [8], *i.e.,* it converges at the same result no matter what rules in Σ are used and in what order the rules are applied. The errors fixed are logical consequences of the rules and ground truth; as long as the rules and ground truth are correct, and if ML predicates in REE^{++}s are accurate, the fixes are correct, known as *certain fixes* [38]. Rock corrects errors in batch and incremental modes; both algorithms are parallelly scalable.

4 A UNIFORM PROCESS

This section presents how Rock identifies tuples (ER), resolves inconsistencies (CR), deduces temporal orders (TD) and imputes missing values (MI) in the same process. We first extend the chase [79] (Section 4.1), and then show how to leverage the interactions of the four and improve the overall data quality via the chase (Section 4.2).

4.1 Chasing with REE^{++}s

To conduct ER, CR, TD and MI uniformly, we mine a set Σ of REE^{++}s and chases the data with the REE^{++}s. Below we give an overview of the chase with REE^{++}s. We start with fixes and ground truth, and then extend the chase [79] with the Church-Rosser property [8].

Fixes. Given a set Σ of REE^{++}s, we apply them to deduce *fixes* in $\mathcal{D}$, maintained in $\overline{U} = (\mathcal{E}_=, \mathcal{E}_\preceq)$. For each tuple in $\mathcal{D}$ with id EID (resp. each A-attribute of EID), a set $[\mathsf{EID}]_=$ (resp. $[\mathsf{EID}.A]_=$) is in $\mathcal{E}_=$, including the ids of entities that are validated to be the same as EID (resp. the constant c such that $\mathsf{EID}.A = c$ is validated). For each attribute A of schema R in $\mathcal{R}$, a set $[A]_\preceq$ is in $\mathcal{E}_\preceq$, including all ranked pairs (t_1, t_2) such that $t_1 \preceq_A t_2$ is validated. Intuitively, the fixes tell us what entities should be identified, what value an attribute should take, and how attribute values are temporally ordered.

Validity. We say that $\overline{U}$ is *valid* if it has no conflicting fixes in $\overline{U}$, *e.g.,* there exist no attribute A, entity id EID and tuples t_1, t_2 such that (a) $[\mathsf{EID}.A]_=$ includes both constants c and d, but $c \neq d$; that is, each attribute has a unique value; and (b) $[A]_\preceq$ includes both ranked pairs (t_1, t_2) and (t_2, t_1), but either $t_1[A] \prec t_2[A]$ or $t_2[A] \prec t_1[A]$.

Ground truth. To justify the correctness of fixes, we only apply an REE^{++} in Σ if its precondition is satisfied by a collection of *"ground truth"*, which is a set $\Gamma = (\Gamma_=, \Gamma_\preceq)$ of validated data, where $\Gamma_=$ (resp. $\Gamma_\preceq$) is enclosed in $\mathcal{E}_=$ (resp. $\mathcal{E}_\preceq$). Typically, $\Gamma_=$ is initialized based on master data or high-quality knowledge bases, and $\Gamma_\preceq$ is initialized with the temporal orders in $\mathcal{D}$ with initial timestamps. Later, the ground truth in Γ is accumulated and expanded with data validated during chasing via possibly user interaction.

The chase. We deduce fixes by chasing $\mathcal{D}$ with REE^{++}s in Σ and

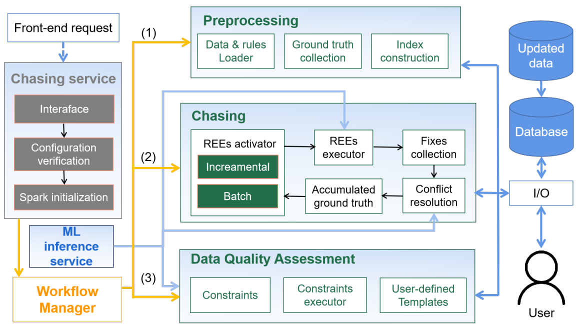


Figure 2: The workflow of chasing

ground truth in Γ. Specifically, the i-th chase step of $\mathcal{D}$ by Σ is:

$$\overline{U}_i \Rightarrow_{(\varphi,h)} \overline{U}_{i+1},$$

where $\varphi : X \rightarrow p_0$ is an REE^{++} in Σ, h is a valuation of φ in $\mathcal{D}$, and the application of (φ, h) should satisfy the following conditions:

(1) All predicates $p \in X$ are *validated* by $\overline{U}$, *e.g.*, if p is $t[A] = c$, then $t[A]$ is validated to be c in $[\text{EID}.A]_=$. If p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, each $A \in \bar{A}$ (resp. $B \in \bar{B}$) is validated in $[\text{EID}.A]_=$ (resp. $[\text{EID}.B]_=$), and $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ = true; similarly for other predicates.

(2) The consequence p_0 extends $\overline{U}_i$ to $\overline{U}_{i+1}$, *e.g.*, if p_0 is $t_1 \preceq t_2$, then add pair (t_1, t_2) to $[A]_\preceq$; similarly for other predicates.

Chasing. A chasing sequence ξ of $\mathcal{D}$ by (Σ, Γ) is

$$\overline{U}_0, \ldots, \overline{U}_k,$$

where $\overline{U}_0 = \Gamma$. For $i \in [1, k]$, there exist REE^{++} φ and valuation h of φ such that $\overline{U}_i \Rightarrow_{(\varphi,h)} \overline{U}_{i+1}$ is a valid chase step, *i.e.*, $\overline{U}_{i+1}$ is valid.

A chasing sequence ξ terminates if either (a) no φ in Σ can be applied; if so, ξ is valid, and $\overline{U}_k$ is its result; or (b) there exist φ, h and $\overline{U}_{k+1}$ such that $\overline{U}_k \Rightarrow_{(\varphi,h)} \overline{U}_{k+1}$ but $\overline{U}_{k+1}$ is invalid (conflict).

Church-Rosser. Following [39, 79], one can verify that the chase is *Church-Rosser* since for any set Σ of REE^{++}s, collection Γ of ground truth, and instance $\mathcal{D}$, all chasing sequences of $\mathcal{D}$ by (Σ, Γ) terminate and converge at the same result, denoted by Chase$(\mathcal{D}, \Sigma, \Gamma)$, no matter what REE^{++}s in Σ are used and in what order they are applied.

Implementing the chase. Although conceptually simple, we cannot directly apply the chase, for two reasons. (a) The enumeration of valuations is costly and worse still, the application of a valuation may rely on other valuations, *e.g.*, not-yet-validated p may become validated after certain chase steps. (b) The chase may terminate at an invalid result. If so, we need to resolve conflicts.

Novelty. To cope with these challenges, we implement the chase in a more efficient way. Its novelty includes the following:

(a) We maintain designated data structures to record temporary chasing results, so that valuations are activated *lazily*.
(b) We develop learning-based strategies to *resolve conflicts*.
(c) We deduce *certain* fixes such that each fix is correct under certain conditions, *i.e.*, it fixes an error and introduces no new errors.

Workflow. As shown in Figure 2, the chasing service is initiated by a front-end request. It starts a process for the distributed work dispatching, scheduling and basic I/O; a workflow manager is used to control the chasing process, which consists of three components: (1) preprocessing, (2) chasing, and (3) data quality assessment.

In the preprocessing step, Rock loads data and rules, initializes the ground truth and builds the data structures for lazy activation.

After preprocessing, Rock proceeds to the chasing step. An REE^{++} activator triggers REE^{++}s so that only valuations that deduce *unknown* fixes are generated. A fix is unknown if it is neither in the ground truth nor has not been deduced before. We support both batch and incremental modes for REE^{++} activation. In the incremental mode, an REE^{++} $\varphi : X \rightarrow p_0$ is activated if at least one predicate in X is validated by the updated data; in the batch mode, as long as there exist predicates in X that are validated by the ground truth, this REE^{++} is activated. The activated REE^{++}s are then forwarded to an executor to perform the chase steps. The fixes deduced are maintained in a collection, whose validity is periodically checked. If there are conflicts in the fixes, we resolve the conflict (see Section 4.2). Otherwise, we accumulate the deduced fixes as new ground truth (possibly with user verification), which may in turn activate more REE^{++}s and the chasing process continues.

Rock adopts built-in constraints and user-defined templates to monitor data quality in terms of completeness, timeliness, validity and consistency [3], *e.g.*, checking nulls/duplicates in an attribute.

4.2 Deduction with the Chase

Below we first present the designated types of REE^{++}s for each of ER, CR, TD and MI. Then we outline how they interact with each other.

ER + CR. To conduct ER, we employ REE^{++}s in Σ with their consequences in the form of t.EID = s.EID or t.EID $\neq$ s.EID, for checking the entities that t and s represent. To conduct CR, we check whether attribute values *violate* the data regularity, where such regularity is modeled by enforcing another type of REE^{++}s in Σ, whose consequences have the form $t.A \oplus c$ or $t.A \oplus s.B$. Specifically, given an REE^{++} $\varphi : X \rightarrow p_0$, a *violation* of φ in $\mathcal{D}$ is a valuation h of φ such that $h \models X$ but $h \not\models p_0$, *i.e.*, h witnesses that $\mathcal{D} \not\models \varphi$. The goal of CR is to identify all such violations and correct the values if needed.

Example 4: Some commodities are sold limited in a special store. Model $\mathcal{M}_{\text{limited}}(t[\text{com}], s[\text{com}])$ checks whether two commodities are sold limited in the same store. REE^{++}s below can be used for ER and CR, by embedding $\mathcal{M}_{\text{limited}}$ as an ML predicate.

φ_9 : $\text{Trans}(t) \land \text{Trans}(s) \land \mathcal{M}_{\text{limited}}(t[\text{com}], s[\text{com}]) \rightarrow t.\text{sid} = s.\text{sid}$. Since the commodities are sold limited in the same store (checked by $\mathcal{M}_{\text{limited}}$), REE^{++} φ_9 identifies the two store ids.

φ_{10} : $\text{Trans}(t) \land \text{Trans}(t') \land \text{Store}(s) \land \text{Store}(s') \land t.\text{sid} = s.\text{sid} \land t'.\text{sid} = s'.\text{sid} \land \mathcal{M}_{\text{limited}}(t[\text{com}], t'[\text{com}]) \rightarrow s.\text{type} = s'.\text{type}$. This rule conducts CR across *two* tables; it states that the same commodity must be sold in the same type of stores. □

TD. We adopt REE^{++}s $X \rightarrow p_0$, where p_0 is $t \preceq_A s$ or $t \prec_A s$. Here X can be either (1) a conjunction of predicates of Section 2.1 and possibly temporal predicates, or (2) $R(t_1) \land R(t_2) \land \mathcal{M}_{\text{rank}}(t_1, t_2, \otimes_A) \rightarrow t_1 \otimes_A t_2$, where $\mathcal{M}_{\text{rank}}$ is an ranking model [42] and $\otimes_A$ is $\prec_A$ or $\preceq_A$; it returns true if it predicts $t_1 \otimes_A t_2$ and false otherwise.

The ranking model $\mathcal{M}_{\text{rank}}$ is trained under a creator-critic framework, by interleaving learning and verification with currency constraints [42]. The creator ranks the temporal orders via $\mathcal{M}_{\text{rank}}$ on attribute values, followed by the critic that validates the ranking and deduces more ranked pairs. The critic produces augmented training data for $\mathcal{M}_{\text{rank}}$ to improve its ranking. Here $\mathcal{M}_{\text{rank}}$ is trained by arranging values chronologically by their distances to a target in the embedding space, and using the distance to quantify the timeliness.

Example 5: Besides REE^{++}s $\varphi_4 - \varphi_6$ in Example 2, we can use the following REE^{++} to deduce timeliness with the ranking model:

φ_{11} : $\mathsf{Person}(t) \wedge \mathsf{Person}(s) \wedge \mathcal{M}_{\mathsf{rank}}(t, s, \preceq_{\mathsf{LN}}) \rightarrow t \preceq_{\mathsf{LN}} s$, which deduces the currency of LN-attribute values based on $\mathcal{M}_{\mathsf{rank}}$. □

MI. When imputing missing values, we combine logic, ML predictions and data extraction from knowledge graphs. Specifically, we use the following three types of REE^{++}s, prioritizing the first two.

(1) (Logic) REE^{++}s of the form $X \rightarrow t[A] = c$, where precondition X may use ML predicates $\mathcal{M}_c \geq \delta$ to assess the correlation between attribute values, *e.g.,* $R(t) \wedge \mathcal{M}_c(t[\bar{A}], t[B] = c) \geq \delta \rightarrow t[B] = c$; intuitively, it says that if $t[\bar{A}]$ and the value c in $t[B]$ are strongly correlated (checked by $\mathcal{M}_c$), then we assign the value c to $t[B]$.

(2) (Data extraction) REE^{++}s: $R(t) \wedge \mathsf{vertex}(x, G) \wedge \mathsf{HER}(t, x) \wedge \mathsf{match}(t[B], x.\rho) \rightarrow t[B] = \mathsf{val}(x.\rho)$. Intuitively, if t matches a vertex x in knowledge graph G and if x reaches vertex v via path ρ (encoding the B-attribute of t), then $t[B]$ takes the value (label) of v.

(3) (ML prediction) REE^{++}s of the form $R(t) \wedge \mathsf{null}(t[B]) \rightarrow t[B] = \mathcal{M}_d(t[\bar{A}], B)$, where $t[\bar{A}]$ is a partial tuple with all validated values and $\mathcal{M}_d$ suggests a value to fill in null $t[B]$ (checked by $\mathsf{null}(t[B])$).

Example 6: REE^{++}s in Example 3 conduct missing value imputation. Another simple REE^{++} that derives the area code is as follows:

φ_{12} : $\mathsf{Store}(t) \wedge t.\mathsf{location} = \text{“Beijing”} \rightarrow t.\mathsf{area\_code} = \text{“010”}$. □

The model $\mathcal{M}_c$ takes a partial tuple $t[\bar{A}]$ and an attribute value $t[B]$ of t ($B \notin \bar{A}$) as input, and returns the confidence of correlation between $t[\bar{A}]$ and $t[B]$ [35]. It is implemented by first pretraining graph embeddings on knowledge graphs, and then generating the confidence by combining the classifications from graph embeddings and language model embeddings. To extend $\mathcal{M}_c$ to $\mathcal{M}_d$, which predicts a value for $t[B]$, we reuse the encoders in $\mathcal{M}_c$ for computing embeddings. It is implemented by first retrieving a set of candidate values for $t[B]$ from graph G based on the partial tuple $t[\bar{A}]$, and then using a ranking model to get a suggested value for $t[B]$.

Interactions. ER, CR, MI and TD interact with each other.

Example 7: Consider the e-commerce database in Tables 1-3.

(1) ER helps CR. Consider a valuation $h_1 = \{(t_{12}, t_{13}) \mapsto (t, s)\}$ of φ_1 that maps tuples t_{12} and t_{13} in D_3 to the tuple variables t and s of φ_1, respectively. By applying (φ_1, h_1), p_1 and p_2 are identified as the same person. Given this, we can correct the erroneous address of p_2, by φ_{13} : $\mathsf{Person}(t) \wedge \mathsf{Person}(s) \wedge t.\mathsf{pid} = s.\mathsf{pid} \wedge X \rightarrow t.\mathsf{home} = s.\mathsf{home}$ where $X = \bigwedge_{A \in \mathcal{T}} t.A = s.A$ and $\mathcal{T}$ is a set of designated attributes such as marital status, salary and the number of kids (not shown). Here φ_{13} is learned from the data; intuitively, the home address of a person is usually unchanged, if the marital status, salary and the number of kids are unchanged. Applying valuation $h_{13} = \{(t_1, t_2) \mapsto (t, s)\}$ of φ_{13}, we fix $t_2[\mathsf{home}]$ =“5 Beijing West Road”.

(2) CR helps TD. Once the errors in home addresses are fixed, we can rank the timeliness of values in attribute home for tuples in D_1. For example, by φ_4 and φ_5 in Example 2, we deduce $t_3[\mathsf{home}]$=“12 Beijing Road” as the current home address for Christine.

(3) TD helps MI. An REE^{++} φ_{14} : $\mathsf{Person}(t') \wedge \mathsf{Person}(t) \wedge \mathsf{Person}(s) \wedge t'.\mathsf{pid} = t.\mathsf{pid} \wedge t.\mathsf{spouse} = s.\mathsf{pid} \wedge t' \preceq_{\mathsf{home}} t \rightarrow s.\mathsf{home} = t.\mathsf{home}$ helps us fill in the home address of person s by a more recent address of his/her spouse. By instantiating the tuple variables in φ_{14} with tuples t_2, t_3 and t_5 in Table 1, respectively, we can impute the missing address for p_4 of George, since George and Christine are married and their home address should be the same.

(4) MI helps ER. After knowing the home address of p_4 is “12 Beijing Road” by (φ_{14}, h_{14}), we can now apply another REE^{++} φ_{15} : $\mathsf{Person}(t) \wedge \mathsf{Person}(s) \wedge t.\mathsf{LN} = s.\mathsf{LN} \wedge t.\mathsf{FN} = s.\mathsf{FN} \wedge t.\mathsf{home} = s.\mathsf{home} \rightarrow t.\mathsf{pid} = s.\mathsf{pid}$, with valuation $h_{15} = \{(t_4, t_5) \mapsto (t, s)\}$ to identify p_3 and p_4 since they have the same name and address. □

Resolving conflicts. Rock resolves conflicts as follows.

(1) (ER or CR) When conflicting entity IDs or attribute values are deduced, Rock presents the conflicts to the users for correction, together with the rules and ground truth that identify the conflicts.

(2) (TD) When conflicting temporal orders are deduced, *i.e.,* $t_1 \preceq_A t_2$ and $t_2 \preceq_A t_1$, but $t_2 \prec_A t_1$ or $t_1 \prec_A t_2$, we resolve the conflict by extending the binary classifier $\mathcal{M}_{\mathsf{rank}}$. Specifically, we extend $\mathcal{M}_{\mathsf{rank}}(t_1, t_2, \otimes_A)$ to output a confidence score from 0 to 1, indicating how likely $t_1 \otimes_A t_2$ holds, where $\otimes_A$ is $\prec_A$ or $\preceq_A$. Then we compute two confidence scores for $t_1 \preceq_A t_2$ and $t_2 \preceq_A t_1$, respectively, and the one with a higher confidence is retained.

(3) (MI) If multiple values are deduced for an attribute value $t[B]$, we can adopt $\mathcal{M}_c$ to decide the suitable value. More specifically, we retrieve a set Cand of candidate values to be $\{c_i \mid \exists \varphi \in \Sigma : X \rightarrow t[B] = c_i$ s.t. X is validated$\}$. We assign $c^* = \arg\max_{c \in \mathsf{Cand}} \mathcal{M}_c(t[\bar{A}], c)$ to $t[B]$, where $t[\bar{A}]$ consists of all validated values.

User inspection is optional for (2) and (3). One can verify that the chase extended with this learning-based conflict resolution remains Church-Rosser [35], and the fixes are certain if the rules and ground truth are correct and if the ML models in REE^{++}s are accurate.

5 IMPLEMENTATION

This section details the implementation of Rock, for its data storage, scalability, modules, optimization (Sections 5.1–5.4, respectively).

5.1 Data Storage and Management

For efficiency and flexibility, Rock built Crystal, a distributed file system to support internet-scale dynamic load across nodes.

Data storage. Crystal develops a consistent hash ring to assign data objects and computing nodes in a cluster to positions in a *virtual* ring structure. It aims to minimize the number of remapped keys when the nodes are updated in the cluster. A standard hashing function CRC-32 [59] is used to encode the IP addresses for hashing the nodes; and data objects are hashed by a self-defined function based on spectral clustering. The mapping between hash codes and nodes are registered in ETCD, a distributed key-value store.

To support data-partitioned parallelism, data objects are partitioned and stored distributedly over a cluster in Crystal. To efficiently fetch objects across nodes, Crystal develops a two-level addressing model. The first-level metadata always resides in the memory of a cluster after the system starts so that each node maintains the global meta information and knows where to fetch data. If a node needs data from the other nodes, it looks up the addressing model and sends messages to the corresponding nodes. Data at each node is partitioned into blocks, stored as a linked list.

Data management. Crystal distributedly stores raw data, metadata, REE^{++}s, ground truth and ML library. It also stores intermediate results produced during rule discovery and rule execution.

Raw data and ground truth management. Crystal loads raw data and correlated ground truth after ETL. Then it pre-processes the data as follows. (1) It adopts built-in rules and regular expressions to address typos and formatting issues. (2) It transforms attribute values to unique ids, and builds (a) a row-oriented copy for the original data, and (b) a column-oriented copy such that similar values are gathered together, via a pretrained built-in model.

Metadata management. Besides the schemas of tables, Crystal also maintains (1) *column distribution*, the distribution of each categorical and numerical attributes; (2) *attribute summary*, a set of signatures for textual attributes, and (3) *external knowledge*, to link suitable ML models and vertices in knowledge graphs to attributes.

ML library and REE^{++}s *management.* Crystal maintains various pretrained models for different tasks and domains. Note that REE^{++}s may embed any existing ML model that returns a Boolean value as predicates. The models are trained by, *e.g.,* data augmentation [67], weak supervision [74], active learning [56] and noisy label handling [81]. In particular, we trained *i.e.,* the ranking model $\mathcal{M}_{\text{rank}}$ for TD and the correlation models $\mathcal{M}_c$ and $\mathcal{M}_d$ for MI.

Rock supports *backend* training for ML models based on feedback from daily work at idle time, to continuously improve the models.

5.2 Scalability

To scale with large data, Rock develops three strategies below.

Load balancing strategy. In rule discovery and error detection/correction [36, 41], each *work unit* is specified as $T = (\varphi, D_T)$, where φ is a (partial) REE^{++} and D_T is a data partition (see below). Statistical information (*e.g.,* support and confidence), detected errors or fixes are returned after T is completed. To ensure load balancing, Rock adopts three simple yet effective strategies:

(1) Data partition: Crystal partitions data into blocks; a work unit may involve multiple blocks. The smaller the blocks, the more units to handle. A good balance would benefit load balancing.

(2) Cost estimation: During work unit generation, Rock estimates the cost of each work unit using the metadata stored in Crystal.

(3) Work unit re-assignment: To scale well with large data, Rock adopts a non-centralized structure under the consistent hash; all nodes in a cluster play the same roles. Each node has its own computing engine and work manager. After all work units are generated, each $T = (\varphi, D_T)$ is distributed to a node based on the hash of D_T. All nodes in the cluster synchronize their status with each other in fixed periods. When a node finishes its assigned work units, it evokes the work manager to fetch work units from other nodes. In this way, Rock achieves load balancing and high scalability; no node is idle unless all work units are finished.

Sampling and top-k strategies. The complexity of rule discovery is inherently exponential. To handle large data, Rock implements the sampling and top-k strategies of [36, 37] to discover top-k REE^{++}s in the samples with the following modifications. (1) Rock samples data with an accuracy guarantee during the discovery pro-

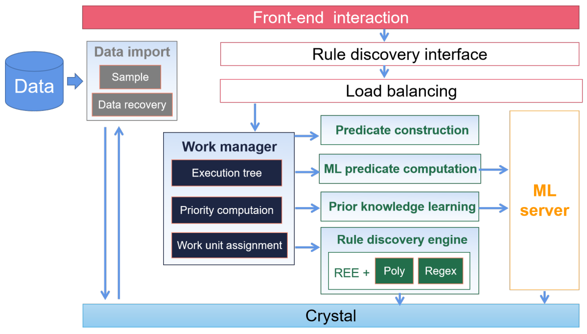


Figure 3: Rule discovery module

cess if the estimated cost of REE^{++} deduction is large. (2) Rock needs users to input their prior knowledge for the top-k discovery, *e.g.,* labels of rules, different applications, and so on. Moreover, Rock (optionally) uses the data coverage as the diversification metric and returns the top-k diversified REE^{++}s. The connection between support and confidence of the rules on samples and their counterparts on the entire dataset is established [36]. Note that discovery with sampling and top-k strategies still returns REE^{++}s that are interpretable.

Prior knowledge learning. Rock supports an interactive process for users to label the usefulness of REE^{++}s and select target predicates for their application, such that only top-k useful REE^{++}s satisfying users' needs are discovered. After a handful of rules are labeled, Rock takes them as training instances, and trains a scoring mode to learn the preferences of users. The model accelerates the rule discovery process. Rock also supports an anytime algorithm to continually return the next top-k results. It iteratively gathers feedback from the users and incrementally trains the model [37].

5.3 Modules

Rock implements the three major modules of Section 3 as follows.

Rule discovery. The module is invoked by a front-end request. As shown in Figure 3, Rock supports a user interface and allows users to confirm the configuration parameters. After the confirmation, a workflow manager is triggered to control (a) data processing, to extract sample data and recover prior intermediate results (if any); (b) predicates, to construct predicates and corresponding auxiliary structures; (c) ML predicates, to compute ML predictions; (d) knowledge learning, to learn prior knowledge and preference from users; (e) rule discovery engine, to mine/learn REE^{++}s in the data; and (f) result management, to check and sort discovered REE^{++}s based on a trained criteria; it also stores intermediate results.

Error detection. This module supports two modes, *batch* and *incremental*. For data-partitioned parallelism, we extended the HyperCube algorithm [41] to divide data into *virtual blocks* and generate work units for each block. The work units are distributed via the consistent hashing; idle nodes fetch work units in real-time from other nodes. The process is controlled by the work manager at each node.

Error correction. This module uniformly conducts ER, CR TD and MI in parallel, by chasing the data with a set Σ of REE^{++}s and a collection of ground truth. It also extends the HyperCube algorithm [41] to divide data into multiple *virtual blocks* and generate work units for each block. To support ML models $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, Locality Sensitive Hashing (LSH) is used to generate hash codes [27], such that

$\bar{A}$ and $\bar{B}$ are transformed to new attributes. If $\mathcal{M}(t[\bar{A}], s[\bar{B}])$ = true, then $\mathsf{LSH}(t][\bar{A}]) = \mathsf{LSH}(s[\bar{B}])$ with high probability. To support MI, attributes of vertices in knowledge bases are extracted as keys and treated as new attributes to satisfy $\mathsf{match}(\cdot, \cdot)$ in REE^{++}s. HyperCube is built by incorporating these new attributes.

Rock evenly distributes the work units to computing nodes in a cluster. Each node executes the assigned work units one by one, and fetches data from other nodes when needed. When a node is idle, a work manager transfers work units from other nodes to it for load balancing. To speed up the chase, each node stores partial valuations in its memory and periodically monitors whether fixes generated by other nodes can be used to resume the executions of these partial valuations. If the partial valuations are too large to fit into the memory of a node, Crystal stores them in the disk locally. The process continues until the cluster generates no more fixes.

Local executor. A *work unit* $T = (\varphi, D_T)$ is executed locally at each node. A query optimizer decides the execution order of predicates in the precondition of φ. Then the local executor maintains partial valuations of φ. If a valuation is complete, the deduced result is saved.

5.4 Optimization

Rock develops strategies to improve efficiency and effectiveness.

Polynomial expressions. Rock identifies arithmetical correlations among numerical attributes as follows. (1) A tree-based model, XGBoost, ranks the importance of numerical attributes via self-supervised learning, and prunes irrelevant features (attributes). (2) Feeding the selected features and labels to a predefined polynomial expression with LASSO regularization, it learns a weight for each feature; unimportant features tend to have zero weights.

Optimization for prior knowledge learning. Asking users to rank REE^{++}s is feasible for data quality experts (who have often already accumulated some rules during their years of practice), but it might be hard for novice users. To make it user friendly, Rock designs a user interaction workflow. After a set of REE^{++}s is discovered, Rock picks a small sample dataset as testing data and detects errors in it. Users are invited to confirm whether the errors are unknown true positives. Rock then collects the user feedback, and incrementally trains its ML model (Section 5.2) to rank REE^{++}s.

Moreover, given a target predicate, Rock adopts an unsupervised ML model based on FDX [95] to prune predicate candidates that are not correlated to the target, to speed up rule discovery.

ML predication. It is costly to conduct ML inference (predictions) at runtime during rule discovery. To rectify this, Rock pre-computes the results in advance once the ML predicates are ready. Moreover, given $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, Rock adopts the *filter-and-verify* paradigm such that (a) a blocking algorithm is first evoked to retrieve a candidate set of potentially matching tuple ID pairs, and then (b) it finds the true matching pairs in the candidate set for $\mathcal{M}(t[\bar{A}], s[\bar{B}])$.

Optimization for rule discovery. Rock continually accumulates ground truth from (a) historical repairs of previous data cleaning tasks and (b) manually labeled data of users, so that the rule discovery module could discover rules on cleaner data. It also embeds various learning-based strategies in the rule discovery engine, including data augmentation, feature generation, and feature enrichment so that the discovered REE^{++}s are more accurate and robust.

6 APPLICATION AND EVALUATION

This section showcases real-world applications of Rock (Bank, Logistics and Sales) for rule discovery (RD), error detection (ED) and error correction (EC), and evaluates its accuracy, efficiency and scalability. More details can be found from the Web page [6].

Baselines. Rock was implemented in Golang. We compared Rock with the following baselines: (1) ES, a rule discovery system that uses the idea of *evidence set* [72] to discover REE^{++}s in parallel in a purely mining manner; (2) T5$_s$ [20], a state-of-the-art ML model based on the pre-trained language model; (3) SparkSQL [14], a data processing module in Apache Spark; (4) Presto [80], a fast and reliable SQL engine for data analytics and the Open lakehouse; (5) RB [65], a holistic data cleaning system that adopts the feature engineering and learns ML models for error detection and correction. No baselines are designed to support all modules (RD, ED and EC), *e.g.,* SparkSQL and Presto do not discover rules/SQL themselves. Thus we only compared a baseline whenever it is feasible.

We also compared Rock with its three variants: (1) Rock$_{\mathsf{noML}}$ that is Rock without ML predicates; (2) Rock$_{\mathsf{seq}}$ that iteratively executes ER, CR, MI and TD one by one until no more changes, and (3) Rock$_{\mathsf{noC}}$ that sequentially conducts ER, CR, MI and TD once.

Real-world applications. We tested three real-life applications.

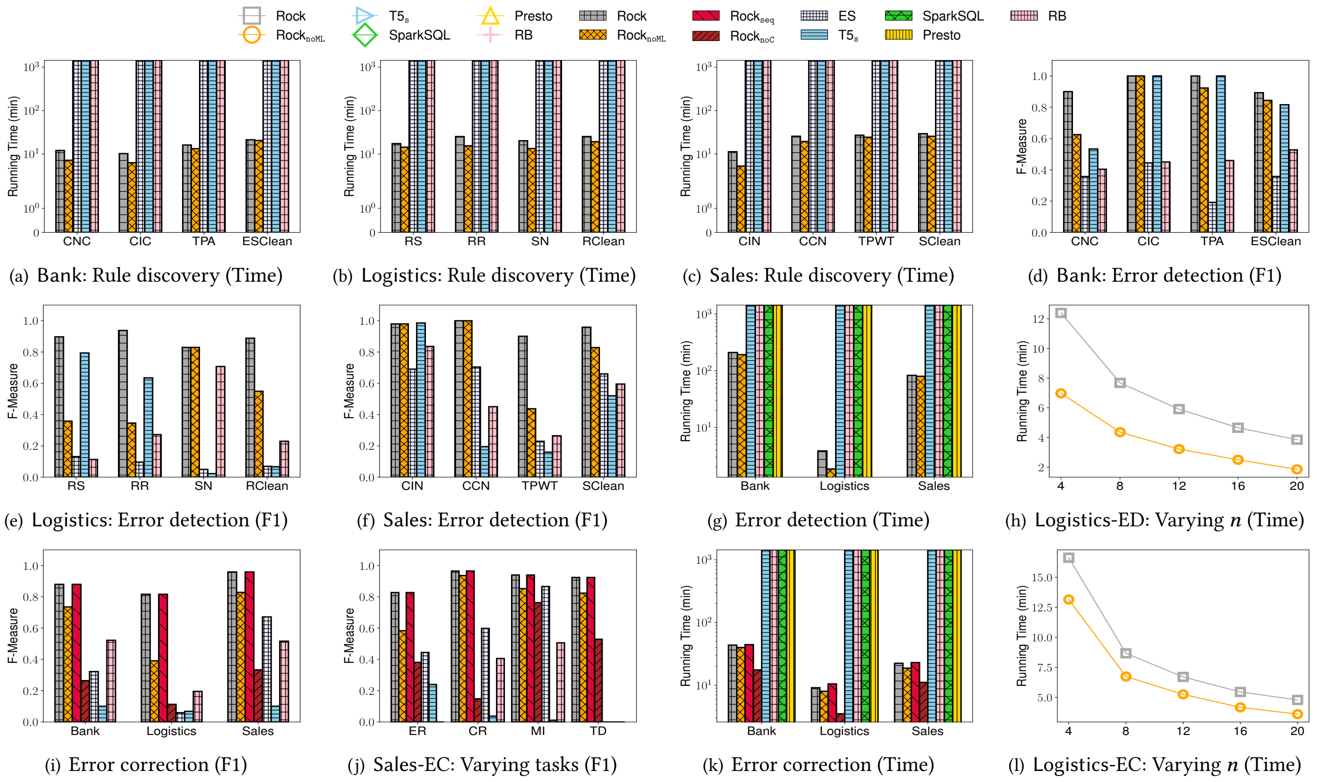
Bank. Nowadays dirty data is one of the top challenges confronted by banks for risk management. A bank needs high-quality data to understand their market and customers, and thus employs Rock. We evaluated a private bank data with 11 relational tables with 1.5 billion tuples and 133 attributes, and report the following four tasks: (a) CNC that cleans names of records in Bank; (b) CIC for company information; (c) TPA that detects and corrects total payment amounts, and (d) ESClean for cleaning all the errors above.

Logistics. A top-tier logistics company has a large volume of logistics data from various domains, *e.g.,* addresses from all over the world, user profiles and order information. However, the poor data quality hampers the value of its data. Rock enhances its data quality to improve its downstream applications. Here we tested one commercial dataset with 1 table and 16 millions of tuples. Four tasks were evaluated: (a) RS for the street information of recipients, (b) RR for cleaning the residential area of recipients, (c) SN that cleans seller names, and (d) RClean for cleaning all the errors above.

Sales. ERP systems need data quality systems to regulate the consistency of sales data. We tested a private commercial dataset of an ERP system with 13 tables, 0.62 billions tuples and 117 attributes with four tasks: (a) CIN that cleans customer information; (b) CCN for company names; (c) TPWT that detects/corrects prices of commodities without tax, and (d) SClean for cleaning all the errors above.

All the tests were conducted on a Kubernetes cluster with 21 virtual nodes, where each node was configured with a 32-core Intel(R) Platinum CPUs at 2.1GHz with 256GB memory. The nodes were connected with a network at 10Gbps. All the experiments were repeated three times and the average is reported here.

Exp-1. Rule discovery (RD). We evaluated the efficiency of rule discovery vs. Rock$_{\mathsf{noML}}$, ES, T5$_s$ and RB. We report the overall time of ES and the training time of T5$_s$ and RB for a fair comparison.



(a) Bank: Rule discovery (Time) (b) Logistics: Rule discovery (Time) (c) Sales: Rule discovery (Time) (d) Bank: Error detection (F1)

(e) Logistics: Error detection (F1) (f) Sales: Error detection (F1) (g) Error detection (Time) (h) Logistics-ED: Varying n (Time)

(i) Error correction (F1) (j) Sales-EC: Varying tasks (F1) (k) Error correction (Time) (l) Logistics-EC: Varying n (Time)

Figure 4: Performance evaluation

The effectiveness of the mined REE^{++}s will be shown later.

We set the support (resp. confidence) threshold as 1e-8 (resp. 0.9) in Rock, and adopted pre-trained ML predicates in our ML pool, *e.g.,* an address normalization model $\mathcal{M}_{\text{addr}}$ based on Bert-CRF, and a commodity SKU identification model $\mathcal{M}_{\text{SKU}}$. We set Rock to discover all relevant REE^{++}s instead of top-k. We set the sampling ratio $r = 10\%$ when the estimated discovery costs are large (Section 5). We extracted 10% tuples from each dataset for training and used the remaining for testing. Note that Rock$_{\text{noML}}$ and ES adopted the same configuration as Rock, and T5$_{\text{s}}$ and RB were fine-tuned based on the validation data.

As shown in Figures 4(a)-4(c), Rock consistently outperforms the competitors (except Rock$_{\text{noML}}$) in efficiency. ES, T5$_{\text{s}}$ and RB cannot finish rule discovery or model training within one day because (a) ES does not have effective pruning strategies; (b) T5$_{\text{s}}$ has to tune millions of parameters, and (c) RB needs costly feature engineering. In contrast, Rock is fast, *e.g.,* it takes 25.2 minutes on average. Rock accelerates the discovery process by pruning a large amount of search space and verifying REE^{++}s in small sample data when the discovery cost is large. Rock$_{\text{noML}}$ is faster than Rock because it does not handle ML predicates; however, its accuracy degrades (see Figure 4(d)-4(f)). Rock returns 388, 47 and 167 REE^{++}s for the three applications, respectively. The rules confirmed by the users were employed for error detection and error correction.

Exp-2. Error detection (ED). To evaluate the accuracy, following [65], we detected errors and manually checked 10,000 tuples for the correctness. We report the F-Measure $= 2 \times \frac{\text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$, where precision (resp. recall) is the ratio of correctly detected errors to all detected errors (resp. to all errors). Rock, Rock$_{\text{noML}}$ and ES identify errors using mined REE^{++}s, while T5$_{\text{s}}$ (resp. RB) uses the generative model (resp. feature engineering and the downstream random forest model) to predict whether values are erroneous. Since ES, T5$_{\text{s}}$ and RB cannot scale to large datasets, we sampled a small amount of data as the training set so that they could finish training in one day.

Accuracy. As reported in Figure 4(d)-4(f), Rock outperforms all baselines, *e.g.,* the average F-Measure of Rock is 60%, 25.8% and 48.9% higher than ES, T5$_{\text{s}}$ and RB, respectively. In particular, when there are many numerical attributes, *e.g.,* Sales, T5$_{\text{s}}$ does not do well, *e.g.,* its F-Measure is 0.52, while it is 0.96 for Rock. This verifies that REE^{++}s are powerful enough to fit the needs of various applications. ES dost not perform well because it is a mining algorithm that mainly focuses on the precision and does not optimize the recall.

By embedding well-trained ML models as predicates, Rock is more accurate than Rock$_{\text{noML}}$, *e.g.,* up to 46.3% for TPWT. ML models could extract relevant information for REE^{++}s, *e.g.,* an address normalization model extracts various associated features including district, city and province from the address attribute of Sales so that Rock discovers three more valuable REE^{++}s than Rock$_{\text{noML}}$. This verifies the need for unifying ML and logic deduction.

Efficiency. We also evaluated the efficiency of Rock, Rock$_{\text{noML}}$, T5$_{\text{s}}$, RB, SparkSQL and Presto for error detection. For a fair comparison, we transformed the learned REE^{++}s to SQL and fed them into SparkSQL and Presto, where ML predicates in REE^{++}s are re-written as UDFs and embedded in SQL. As shown in Figure 4(g), Rock outperforms all baselines (except $\text{Rock}_{\text{noML}}$) over all applications. All baselines, *e.g.,* SparkSQL and Presto, cannot finish the execution in one day, since they support no designated strategy for accelerating ML models. Although REE^{++}s embed ML models as predicates, Rock adopts a blocking strategy (Section 5) and hence, its ML computation does not substantially increase the cost. Moreover, Rock is faster than the existing SQL engines, by extending the HyperCube [41] and adopting strategies to avoid load imbalance.

Parallel scalability. In Figure 4(h) we varied the number n of workers used. One can see that Rock is 3.36× faster when n is changed from 4 to 20, verifying that the error detection of Rock is parallelly scalable, by adding more computing nodes when needed.

Exp-3. Error Correction (EC). In the same setting as Exp-2, we tested Rock with mined REE^{++}s for correcting the detected errors in each dataset. It starts with 10,000 tuples that were manually selected, checked and treated as initial ground truth for Bank, Logistics and Sales. During the chase, an REE^{++} is applied only if its precondition is validated by the ground truth (recall this from Section 4).

Accuracy. As shown in Figure 4(i), Rock outperforms ES, T5_s and RB by 53.4%, 90.9% and 47.3% in F-Measure, respectively. This verifies that chasing with REE^{++}s and accumulated ground truths in Rock makes a promising approach for correction. In particular, T5_s (resp. RB) is not effective for numerical (textual) values, *e.g.,* with 0.10 (resp. 0.52) F-Measure for such attributes, in contrast to 0.96 (resp. 0.88) for Rock. This verifies the advantages of Rock against ML and holistic approaches. We did not compare with Presto and SparkSQL here since they use the same REE^{++}s discovered by Rock.

Moreover, we find the following in an ablation study.

(1) ML predicates. Rock is more accurate than $\text{Rock}_{\text{noML}}$ in the three applications, *e.g.,* $\mathcal{M}_{\text{addr}}$ extracts the information of street, city and province from a single address, and $\mathcal{M}_{\text{SKU}}$ classifies a commodity to a suitable category. Moreover, Rock gives explanation for certain ML models that are treated as consequences of REE^{++}s. These further justify the need for unifying ML and logic rules in Rock.

(2) Interaction. Rock has much higher F-Measure than Rock_{noC}, *e.g.,* 88.5% vs. 23.7% on average. This verifies that supporting interactions among ER, CR, MI and TD indeed helps each other to fix more errors and thus improves the overall data quality, by conducting ER, CR, MI and TD in the same process. Rock has the same F-Measure as Rock_{seq} because both adopt the chasing procedure.

In addition, we evaluated each of ER, CR, MI and TD. As shown in Figure 4(j), Rock consistently beats all baselines for all tasks, *e.g.,* 44.7% and 58.8% more accurate than Rock_{noC} and T5_s for ER, respectively. That is, Rock improves the accuracy of each individual task. Note that TD of ES, TD of T5_s, TD and ER of RB are not shown because they do not support these operations.

Efficiency. We tested Rock, $\text{Rock}_{\text{noML}}$, Rock_{noC}, Rock_{seq}, RB, T5_s, Presto and SparkSQL in efficiency. To simulate the chase of Rock, we iteratively executed SQL in SparkSQL and Presto, and ran ML inference of T5_s and RB until no more fixes can be generated.

As shown in Figure 4(k), Rock is the fastest (except $\text{Rock}_{\text{noML}}$and Rock_{noC}), *e.g.,* it is at least 33× faster than SparkSQL and Presto, partially due to its partial valuation, blocking techniques for ML predicates and load balancing. While T5_s only scans the datasets once, its transformer uses a large number of parameters and thus, is costly in inference. RB is relatively slow, and cannot finish error correction in one day, since its feature generation is costly.

Note that Rock_{noC} is faster than Rock and Rock_{seq} since Rock_{noC} only executes ER, CR, MI and TD once while both Rock and Rock_{seq} run until the chase terminates. Also observe that Rock is faster than Rock_{seq}, *e.g.,* the average runtime of Rock and Rock_{seq} is 29 and 32 minutes, respectively. This is because Rock can select any suitable REE^{++} to execute in each iteration regardless of ER, CR, MI and TD, while Rock_{seq} blindly tries each REE^{++} of the four tasks one by one. This verifies that conducting the four tasks in the same process is no more costly than sequential execution (Section 4.2).

Scalability. We also varied the number n of workers in Figure 4(l). Rock is parallelly scalable; it is 3.12× faster when n is from 4 to 20.

Exp-4. Real-life evaluation. Below are what clients reported, about how Rock improved the performance of their applications.

Bank. Rock cleaned the dataset of a top-tier commercial bank. To effectively improve the data quality, domain experts injected their business knowledge into Rock, and Rock iteratively cleaned the data. In each round, Rock executed the rule discovery module to discover a set of rules from the (dirty) data. These rules were fed to the error detection module. The detected error were returned to the domain experts, who were invited to label whether (selected) errors are true positives. The labeled data was added to ground truth, to refine rule learning. When a small amount ground truth was accumulated, Rock evoked the error correction module to fix the errors. The iteration ended when no more fixes were generated.

According to our bank client, Rock improves the F-Measure of their data cleaning system on a labeled dataset from 80.1% to 97.7%. Rock also reduces manual efforts of customer confirmations by 8×.

Logistics. A top-tier logistics company wants schema mapping to link correlated attributes across relational tables. To ensure accurate schema mapping, they employed Rock to clean their data first. Their data is fairly consistent, but is incomplete (with a large number of null values). Hence Rock first discovered and employed REE^{++}s to impute missing values via the chase. Then it evoked the predicate construction component in the rule discovery module to find pairs of correlated attributes. Since there are a large number of tables (20K+), Rock triggered a blocking step such that feature vectors of attributes were generated and only attributes with similar features were kept for further verification. As reported by our client, the F-Measure of Rock is above 85%, much higher than the other methods.

This shows how Rock was used for a designated task MI.

Data cleaning in e-commerce. We show how to improve the accuracy of recommendation from a data cleaning perspective.

The company adopted a recommendation model $\text{deepFM}(x, y)$, where x is in a User table and y is in an Item table. If $\text{deepFM}(x, y)$ predicts true, it recommends y to x [19, 98]. Typically, additional features are extracted for x and y from external sources crawled and accumulated (*e.g.,* external user data in UserExt and item data in ItemExt), to improve the accuracy of $\text{deepFM}(x, y)$. The external tables UserExt and ItemExt are often quite dirty and lack labeled entities for ER, *e.g.,* there are inconsistencies between product categories and names. To rectify these, Rock adopts a bootstrapping strategy [55], such that it iteratively executes rule discovery and accumulates clean data until no more rules can be discovered. Below are a few sample REE^{++}s that Rock has mined and applied.

(1) (ER) φ_{ER} : $\text{Item}(t) \wedge \text{ItemExt}(s) \wedge t.\text{Cat} = s.\text{Cat} \wedge \mathcal{M}_{\text{ER}}(t[\bar{A}], s[\bar{B}]) \rightarrow t.\text{id} = s.\text{id}$, where $\bar{A}$ (resp. $\bar{B}$) are all attributes of t (resp. s) and $\mathcal{M}_{\text{ER}}$ is an ER model that identifies entities from the external source. This REE^{++} identifies two items across two tables if they have the same category and the model predicts them as a match.

(2) (CR) φ_{CR} : $\text{ItemExt}(t) \wedge t.\text{name} = \text{“IPhone14”} \rightarrow t.\text{year} = \text{“2022”}$, assuring that the release year of "IPhone14" is "2022".

(3) (TD) φ_{TD} : $\text{User}(t) \wedge \text{User}(s) \wedge \mathcal{M}_{\text{rank}}(t, s, \preceq_{\text{latestProduct}}) \rightarrow t \preceq_{\text{latestProduct}} s$, which temporally ranks the latest used products.

(4) (MI) φ_{MI} : $\text{User}(t) \wedge \text{UserExt}(s) \wedge \mathcal{M}_{\text{ER}}(t[\text{id}], s[\text{id}]) \rightarrow t.\text{latestProduct} = s.\text{product}$. This rule imputes the missing product used by user t, by referencing the value of tuple s in the external source, if t and s are identified via the ER model $\mathcal{M}_{\text{ER}}$ above.

Then Rock evokes the error correction module to conduct the chase for ER, CR, MI, and TD. Taking a user $t = (\text{name} = \text{John}, \text{latestProduct} = \text{null}, \text{boughtYear} = 2021, \ldots)$ and an item $s = (\text{name} = \text{IPhone14}, \text{cat} = \text{mobile}, \text{year} = \textit{2002}, \ldots)$ as an example, deepFM may not accurately decide whether user t will buy item s, due to the lack of information (*e.g.,* t.latestProduct is null) and the erroneous values (*e.g.,* s.year is wrong). In contrast, by chasing the data with the REE^{++}s above, Rock can fix the errors and impute the missing values, *e.g.,* t.latestProduct is filled by "IPhone13" via φ_{MI} and s.year is corrected to be "2022" via φ_{CR}. Then it is reasonable to recommend s to t since the latest product bought by t is "IPhone13", which is an earlier series of s, and this user-item pair (t, s) can serve as a new positive example for (incrementally) training deepFM.

Rock can also employ an REE^{++} to directly enrich deepFM.

(5) (Enrich) φ_{Enrich} : $\text{User}(t) \wedge \text{Item}(s) \wedge \text{deepFM}(t, s) < \delta \wedge X_1 \rightarrow p_0$, where X_1 adds logic conditions (*e.g.,* t recently receives a coupon for s), and p_0 is $t.\text{recommendedItem} = s.\text{id}$ (*i.e.,* we recommend s to t). That is, although deepFM(t, s) predicts that user t may not like item s (with confidence below threshold δ), if X_1 holds, then we override the prediction of deepFM and recommend s to t.

Summary. We find the following. (1) In real-life applications, Rock outperforms the state-of-the-art systems for rule discovery, error detection and correction. It is 25.8% (resp. 47.3%) more accurate than the best baseline on average for error detection (resp. correction), and is up to 374× (resp. 157×) faster. (2) Rock performs the best on all ER, CR, TD and MI, *e.g.,* its F-measure on CR and ER is as high as 0.965 and 0.828, respectively. (3) By embedding ML predicates, Rock improves the F-Measure by 20.5% on average, up to 59.2%. (4) Rock could find 388, 47 and 167 high-quality REE^{++}s for the three applications, respectively. (5) As evaluated by our clients, Rock is accurate, *e.g.,* 85% F-Measure by our Logistics client.

7 RELATED WORK

Systems. We categorize systems related to Rock as follows.

ER+CR. In industry, data management products offer integration solutions, *e.g.,* Talend [5], AmazonGlue [2], Informatica [4] and Ataccama [1] integrate data from different sources. There have also been ER systems from academy, *e.g.,* DADER [85], JedAI [70] and Magellan [58] that adopt rules and ML models to match entities. Besides, Talend [5], Informatica [4] and Ataccama [1] provide CR solutions. Data cleaning systems from academy, *e.g.,* BigDansing [57], CoClean [69], Horizon [77], NADEEF [26], CODED [90] and SCODED [91], detect conflicts and repair data via logical/statistical methods. AlphaClean [61] tunes hyperparameter for cleaning.

TD. Only a few systems support temporal deduction [7], and temporal data management [64, 94]. In particular, Tamr [7] adopts rules and patterns to identify the best records of an entity.

MI. Informatica [4], Ataccama [1] and Tamr [7] provide data enrichment solutions. Saga [53] is a serving platform for knowledge enrichment in various applications. ActiveDeeper [97] and KGLac [49] from academy inject external knowledge to enrich data.

Different from the prior systems, Rock (1) provides a uniform framework for logic rules and ML with REE^{++}s, to take advantage of both and provide logic interpretation under certain conditions; (2) supports ER, CR, TD and MI in the same process, (3) warrants the corrections to errors as logical consequences of REE^{++}s and accumulated ground truth when the embedded ML models are accurate, (4) supports the parallel scalability to scale with large datasets in principle, (5) learns users' prior knowledge and discovers rules under both subjective and objective measures, and (6) enhances the ability for data cleaning across multiple relational tables. In particular, (7) for TD, Rock employs a creator-critic framework to deduce currency. (8) For MI, it integrates logic, ML prediction and data extraction from knowledge graphs for data enrichment, and it continuously maintains (resp. trains) the knowledge graphs (resp. ML models).

Machine learning. Various ML models have been employed for data cleaning, *e.g.,* Ditto [63], DeepMatcher [68], DADER [84], AutoEM [96], ChatGPT based method [71] for ER; HoloClean [76], HoloDetect [48], Raha [66], RetClean [9] for CR; ranking models (see [89] for a survey) for TD; and denoising autoencndoer [86], GAN [92] and attention mechanism [83] for MI.

Rock can embed these models as predicates in REE^{++}s if their outputs are transformed to Boolean values, *e.g.,* by referencing a threshold. It supports all ER, CR, TD and MI in a unified process.

8 CONCLUSION

Dirty data remains a clear and present danger to big data analytics. Rock aims to (a) unify logic deduction and ML, (b) improve the overall quality by catching and fixing duplicates, conflicts, missing data and obsolete values in the same process, (c) improve the accuracy of its fixes to errors, and (d) scale with large datasets.

We plan to extend Rock and support the following: (a) federated learning across multiple private data sources; (b) more user-friendly methods to learn from users' prior knowledge; and (c) effective algorithms to learn top-k diversified rules, such that on the one hand, the rules are as close to users' interest as much as possible, and on the other hand, they are as diverse to each other as possible.

ACKNOWLEDGMENTS

Dr. Yaoshu Wang is supported by Longhua Science and Technology Innovation Bureau 10162A20220720B12AB12. Dr. Min Xie is supported in part by China NSFC 62202313 and Guangdong Basic and Applied Basic Research Foundation 2022A1515010120.

REFERENCES

[1] 2023. Ataccama: Unified Data Management Platform. https://www.ataccama.com/.

[2] 2023. AWS Glue: Discover, prepare, and integrate all your data at any scale. https://aws.amazon.com/glue/.

[3] 2023. Data Governance DataArts Studio. https://support.huaweicloud.com/usermanual-dataartsstudio/dataartsstudio_01_0715.html?version=2.5.50000.157&platform=win.

[4] 2023. Informatica: Data chaos becomes data clarity. https://www.informatica.com/.

[5] 2023. Modern data management that drives real value. https://www.talend.com/.

[6] 2023. Rock. http://www.grandhoo.com/en.

[7] 2023. Tamr: Next-Generation Data Mastering & Enrichment. https://www.tamr.com/.

[8] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[9] Mohammad Shahmeer Ahmad, Zan Ahmad Naeem, Mohamed Y. Eltabakh, Mourad Ouzzani, and Nan Tang. 2023. RetClean: Retrieval-Based Data Cleaning Using Foundation Models and Data Lakes. *CoRR* abs/2303.16909 (2023).

[10] João Paulo Aires and Felipe Meneguzzi. 2017. Norm Conflict Identification Using Deep Learning. In *AAMAS Workshops*. 194–207.

[11] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.

[12] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*. 952–963.

[13] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[14] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational Data Processing in Spark. In *SIGMOD*. ACM, 1383–1394.

[15] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* 52, 3 (2013), 441–482.

[16] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *TKDD* (2007).

[17] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *SIGMOD*. ACM, 143–154.

[18] Statistics Canada. 2022. Classification of legal marital status. https://www23.statcan.gc.ca/imdb/p3VD.pl?Function=getVDn&TVD=61748&CVD=61748&CLV=0&MLV=1&D=1.

[19] Xu Chen, Yongfeng Zhang, and Zheng Qin. 2019. Dynamic Explainable Recommendation Based on Neural Attentive Models. In *AAAI*. AAAI Press, 53–60.

[20] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. https://doi.org/10.48550/ARXIV.2210.11416

[21] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *TODS* 4, 4 (1979), 397–434.

[22] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. 315–326.

[23] Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling Up Hands-Off Crowdsourced Entity Matching to Build Cloud Services. In *SIGMOD*. 1431–1446.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[25] Mohamad Dolatshah, Mathew Teoh, Jiannan Wang, and Jian Pei. 2018. Cleaning Crowdsourced Labels Using Oracles For Statistical Classification. *PVLDB* 12, 4 (2018), 376–389.

[26] Amr Ebaid, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A Generalized Data Cleaning System. *PVLDB* (2013).

[27] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* 11, 11 (2018), 1454–1467.

[28] Exasol. 2020. Exasol Research Finds 58% of Organizations Make Decisions Based on Outdated Data. *https://www.exasol.com/news-exasol-research-finds-organizations-make-decisions-based-on-outdated-data/*.

[29] Lihang Fan, Wenfei Fan, Ping Lu, Chao Tian, and Qiang Yin. 2024. Enriching Recommendation Models with Logic Conditions. *Proc. ACM Manag. Data* (2024).

[30] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[31] Wenfei Fan, Ling Ge, Ruochun Jin, Ping Lu, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE, 634–647.

[32] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 1 (2008), 25:1–25:49.

[33] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.

[34] Wenfei Fan, Floris Geerts, and Jef Wijsen. 2012. Determining the Currency of Data. *TODS* 37, 4 (2012), 25:1–25:46.

[35] Wenfei Fan, Ziyan Han, Weilong Ren, Ding Wang Yaoshu Wang, Min Xie, and Mengyi Yan. 2024. Splitting Tuples of Mismatched Entities. In *SIGMOD*. ACM.

[36] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. ACM, 384–398.

[37] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2023. Discovering Top-k Rules using Subjective and Objective Criteria. *Proc. ACM Manag. Data* 1, 1 (2023), 70:1–70:29.

[38] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards Certain Fixes with Editing Rules and Master Data. *PVLDB* 3, 1 (2010), 173–184.

[39] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying logic rules and machine learning for entity enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[40] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing Certain Fixes to Graphs. *PVLDB* 12, 7 (2019), 752–765.

[41] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.

[42] Wenfei Fan, Resul Tugay, Yaoshu Wang, Min Xie, and Muhammad Asif Ali. 2023. Learning and Deducing Temporal Orders. *PVLDB* 16, 8 (2023), 1944–1957.

[43] I.P. Fellegi and D. Holt. 1976. A Systematic Approach to Automatic Edit and Imputation. *J. of the American Statistical Association* 71, 353 (1976), 17–35.

[44] Ivan P. Fellegi and Alan B. Sunter. 1969. A Theory for Record Linkage. *J. of the American Statistical Association* 64, 328 (1969), 1183–1210.

[45] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *WWW*. 2331–2341.

[46] Gartner. 2018. How to create a business case for data quality improvement. *https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/*.

[47] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.

[48] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846.

[49] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A Demonstration of KGLac: A Data Discovery and Enrichment Platform for Data Science. *PVLDB* 14, 12 (2021), 2675–2678.

[50] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[51] Rachael A Hughes, Jon Heron, Jonathan AC Sterne, and Kate Tilling. 2019. Accounting for missing data in statistical analyses: multiple imputation is not always the answer. *International journal of epidemiology* 48, 4 (2019), 1294–1304.

[52] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal* 42, 2 (1999), 100–111.

[53] Ihab F. Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed A. Soliman. 2022. Saga: A Platform for Continuous Construction and Serving of Knowledge at Scale. In *SIGMOD*. ACM, 2259–2272.

[54] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk Server - Adding missing Links while consuming Linked Data. In *COLD*, Vol. 665.

[55] Heinrich Jiang and Maya R. Gupta. 2021. Bootstrapping for Batch Active Sampling. In *SIGKDD*. Association for Computing Machinery, 3086–3096.

[56] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. *arXiv preprint arXiv:1906.08042* (2019).

[57] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. BigDansing: A System for Big Data Cleansing. In *SIGMOD*. 1215–1230.

[58] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016), 1197–1208.

[59] Philip Koopman. 2002. 32-Bit Cyclic Redundancy Codes for Internet Applications. In *2002 International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, 459–472.

[60] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1 (2010), 484–493.

[61] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. *CoRR* abs/1904.11827 (2019).
[62] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A Complexity Theory of Efficient Parallel Algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[63] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.
[64] Wei Lu, Zhanhao Zhao, Xiaoyu Wang, Haixiang Li, Zhenmiao Zhang, Zhiyu Shui, Sheng Ye, Anqun Pan, and Xiaoyong Du. 2019. A Lightweight and Efficient Temporal Database Management System in TDSQL. *PVLDB* 12, 12 (2019), 2035–2046.
[65] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *PVLDB* 13, 11 (2020), 1948–1961.
[66] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*. 865–882.
[67] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond. In *SIGMOD*. ACM, 1303–1316.
[68] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*. 19–34.
[69] Mashaal Musleh, Mourad Ouzzani, Nan Tang, and AnHai Doan. 2020. CoClean: Collaborative Data Cleaning. In *SIGMOD*. ACM, 2757–2760.
[70] George Papadakis, George Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional entity resolution with JedAI. *Information Systems* 93 (2020), 101565.
[71] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for Entity Matching. In *New Trends in Database and Information Systems - ADBIS*.
[72] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *PVLDB* 13, 3 (2019), 266–278.
[73] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *CIKM*. 1379–1388.
[74] Alexander J. Ratner, Braden Hancock, and Christopher Ré. 2019. The Role of Massively Multi-Task and Weak Supervision in Software 2.0. In *CIDR*. www.cidrdb.org.
[75] Thomas C. Redman. 2016. Bad Data Costs the U.S. $3 Trillion Per Year. Harvard Business Review. *https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year*.
[76] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.
[77] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *PVLDB* 14, 11 (2021), 2546–2554.
[78] Royal Mail. 2018. Dynamic Customer Data in a Digital World: Data Services Insight Report. *https://www.royalmail.com/business/system/files/royal-mail-data-services-insight-report-2018.pdf*.
[79] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *SIGMOD*.
[80] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on Everything. In *ICDE*. IEEE, 1802–1813.
[81] Hwanjun Song, Minseok Kim, Dongmin Park, and Jae-Gil Lee. 2020. Learning from Noisy Labels with Deep Neural Networks: A Survey. *CoRR* abs/2007.08199 (2020).
[82] Katia P. Sycara. 1993. Machine learning for intelligent support of conflict resolution. *Decision Support Systems* 10, 2 (1993), 121–136.
[83] Simon Tihon, Muhammad Usama Javaid, Damien Fourure, Nicolas Posocco, and Thomas Peel. 2021. DAEMA: Denoising Autoencoder with Mask Attention. In *ICANN (Lecture Notes in Computer Science)*, Vol. 12891. Springer, 229–240.
[84] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Chengliang Chai, Guoliang Li, Ruixue Fan, and Xiaoyong Du. 2022. Domain Adaptation for Deep Entity Resolution. In *SIGMOD*. ACM, 443–457.
[85] Jianhong Tu, Xiaoyue Han, Ju Fan, Nan Tang, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2022. DADER: Hands-Off Entity Resolution with Domain Adaptation. *PVLDB* 15, 12 (2022), 3666–3669.
[86] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* 11 (2010), 3371–3408.
[87] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *SSDBM*. 1:1–1:12.
[88] Steven Euijong Whang and Hector Garcia-Molina. 2013. Joint entity resolution on multiple datasets. *The VLDB Journal* 22, 6 (2013), 773–795.
[89] Jun Xu, Xiangnan He, and Hang Li. 2020. Deep Learning for Matching in Search and Recommendation. *Found. Trends Inf. Retr.* 14, 2-3 (2020), 102–288.
[90] Jing Nathan Yan, Oliver Schulte, Jiannan Wang, and Reynold Cheng. 2019. Detecting Data Errors with Statistical Constraints. *CoRR* abs/1902.09711 (2019).
[91] Jing Nathan Yan, Oliver Schulte, Mohan Zhang, Jiannan Wang, and Reynold Cheng. 2020. SCODED: Statistical Constraint Oriented Data Error Detection. In *SIGMOD*.
[92] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 5675–5684.
[93] Dongxiang Zhang, Long Guo, Xiangnan He, Jie Shao, Sai Wu, and Heng Tao Shen. 2018. A Graph-Theoretic Fusion Framework for Unsupervised Entity Resolution. In *ICDE*. IEEE, 713–724.
[94] Shiming Zhang, Yin Yang, Wei Fan, Liang Lan, and Mingxuan Yuan. 2014. OceanRT: real-time analytics over large temporal data. In *SIGMOD*. ACM, 1099–1102.
[95] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. 2020. A Statistical Perspective on Discovering Functional Dependencies in Noisy Data. In *SIGMOD*. ACM, 861–876.
[96] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*. 2413–2424.
[97] Liang Zhao, Qingcan Li, Pei Wang, Jiannan Wang, and Eugene Wu. 2020. ActiveDeeper: A Model-based Active Data Enrichment System. *PVLDB* 13, 12 (2020), 2885–2888.
[98] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *SIGIR*. ACM, 749–758.