Splitting Tuples of Mismatched Entities

WENFEI FAN, Beihang University, China, Shenzhen Institute of Computing Sciences, China, and University of Edinburgh, United Kingdom
ZIYAN HAN, Beihang University, China
WEILONG REN*, Shenzhen Institute of Computing Sciences, China
DING WANG, Center of Inernet Governance Research of Tsinghua University, China
YAOSHU WANG, Shenzhen Institute of Computing Sciences, China
MIN XIE, Shenzhen Institute of Computing Sciences, China
MENGYI YAN, Beihang University, China

There has been a host of work on entity resolution (ER), to identify tuples that refer to the same entity. This paper studies the inverse of ER, to identify tuples to which distinct real-world entities are matched by mistake, and split such tuples into a set of tuples, one for each entity. We formulate the tuple splitting problem. We propose a scheme to decide what tuples to split and what tuples to correct without splitting, fix errors/assign attribute values to the split tuples, and impute missing values. The scheme introduces a class of rules, which embed predicates for aligning entities across relations and knowledge graphs G, assessing correlation between attributes, and extracting data from G. It unifies logic deduction, correlation models, and data extraction by chasing the data with the rules. We train machine learning models to assess attribute correlation and predict missing values. We develop algorithms for the tuple splitting scheme. Using real-life data, we empirically verify that the scheme is efficient and accurate, with F-measure 0.92 on average.

CCS Concepts: • **Information systems** → **Information integration**.

Additional Key Words and Phrases: Tuple splitting; data quality; entity resolution

ACM Reference Format:
Wenfei Fan, Ziyan Han, Weilong Ren, Ding Wang, Yaoshu Wang, Min Xie, and Mengyi Yan. 2023. Splitting Tuples of Mismatched Entities . *Proc. ACM Manag. Data* 1, 4 (SIGMOD), Article 269 (December 2023), 29 pages. https://doi.org/10.1145/3626763

1 INTRODUCTION

One of the most studied topics of data quality is entity resolution (ER). The ER problem is to identify tuples that refer to the same entity. Also known as record linkage, data deduplication, merge/purge and record matching, ER has been serving as a routine operation in many applications. There has been a large body of work on ER, via machine learning (ML) [24, 40, 46, 58, 86, 88, 99, 105, 124, 131,

*Corresponding author

Authors' addresses: Wenfei Fan, Beihang University, China and Shenzhen Institute of Computing Sciences, China and University of Edinburgh, United Kingdom, wenfei@inf.ed.ac.uk; Ziyan Han, Beihang University, China, hanzy@act.buaa.edu.cn; Weilong Ren, Shenzhen Institute of Computing Sciences, China, renweilong@sics.ac.cn; Ding Wang, Center of Inernet Governance Research of Tsinghua University, China, dingwang@mail.tsinghua.edu.cn; Yaoshu Wang, Shenzhen Institute of Computing Sciences, China, yaoshuw@sics.ac.cn; Min Xie, Shenzhen Institute of Computing Sciences, China, xiemin@sics.ac.cn; Mengyi Yan, Beihang University, China, yanmy@act.buaa.edu.cn.

2836-6573/2023/12-ART269 $15.00
https://doi.org/10.1145/3626763

information from the Swiss director	information from the German director	erroneous values

tid	name	nationality	born	college	film	filmFestival	festCity	festCountry
t_s	Noemi Schneider	Swiss	2013	null	Sturm	DOK.fest	Munich	null
t_c	Noemi Schneider	Japanese	1986	ZHdK	Sturm	Landshut Short Film Festival	Landshut	Germany

Fig. 1. Example tuples of mismatched entities

135], logic rules [25, 30, 48, 66, 123] or hybrids of the two [27, 55, 57].

A related problem concerns splitting tuples of mismatched entities. In practice, distinct entities may be matched to the same tuple by mistake, despite efforts for preventing so (*e.g.,* the study of hard/soft *conflicts* [36, 122, 127], *i.e.,* when an attribute takes multiple values, or two attributes bear values that cannot co-occur; *e.g.,* when the status of a person is assigned both "single" and "married").

Tuples of mismatched entities are observed in (a) user-contri-buted projects (*e.g.,* IMDb [16] and WikiPedia [22]), where users manipulate data collaboratively yet separately, (b) third-party data purchased from for-profit companies [38] (*e.g.,* Dun & Bradstreet [14]) or public data aggregators (*e.g.,* US Bureau of Labor Statistics [20]), where data has been processed by other parties, and (c) data cleaning under data lineage where cleaning starts from checkpoints. For instance, it has been reported in IMDb [129] that two different series (CG series [1] and Web series [2]) with the same title "Lego Friends" were merged into one, since "someone unfortunately made a mistake when creating the entries" [129]. This mismatch was then manually split into two, one for a distinct series. Similar issues were also reported in Wikidata [73], where Joseph de Cambis (Q3185827) was requested to split into two, one for Joseph de Cambis (1658-1736) and one for Joseph de Cambis (1748-1825).

Intuitively, tuple splitting (TS) is the inverse of ER. ER identifies different tuples that denote the same entity and merges them into a single tuple, while TS identifies a tuple of mismatched entities and decomposes it into different tuples to represent distinct entities. As reported by Wikidata [8], "such amalgamation can happen, *e.g.,* when individuals have similar names and are active in related fields". When searching on IMDb community forums with keywords "merge" and "split", it returns 6.7k and 886 results, respectively; after manual inspection, we find that 7 out of the top-10 results for "split" shows actual needs for splitting a mismatch of persons/movies. As another example, it was reported that the number of author splits is around one half of the number of merges in DBLP in 2019 [107]. While TS is not as frequent as ER, a small percentage of mismatches could be quite damaging; as stated in IMDb help center, "incorrect merges can take a long time to correct and during that time the information will be listed badly on the Website" [68].

Unfortunately, while ER has been well studied and many applications support automatic merge (*e.g.,* IOS 16 [33]), the importance of TS is underestimated, and only limited TS functionalities are provided, *e.g.,* IMDb only supports a "Name Split" option to split credits by roles [17], and Wikidata splits tuples manually by moving attributes one by one [8]. Add to the complication that in some applications, we do not know what ER methods were used and what the original data is, *e.g.,* when ER is conducted by other parties in collaborative cleaning or the data is obtained from a third party. If we know how ER merges two tuples, we may use conflicts as an evidence of mismatch and then, revert the wrong merge and improve the ER itself. However, without knowing what/how ER was actually used in the original data, "it is preferable to split the items" [8] and a sophisticated method for splitting tuples is needed.

Example 1: Consider a *real mismatch* in IMDb [47], where two directors named "Noemi Schneider" are merged by mistake. One is a Swiss, born in 1986 in Brugg and studied at ZHdK; the other is a German, born in Munich and studied film there; both were nominated for an award in some film festivals. The merged (simplified) tuple is t_s in Fig. 1; its schema is person = (name, nationality, born, college, film, filmFestival, festCity, festCountry). The erroneous values and the correct belonging of each value of t_s are colored.

A close examination of t_s reveals the following: (a) its film "conflicts" with its filmFestival, *i.e.,* no short film named "Sturm" was ever nominated at "DOK.fest" [13], and (b) "Sturm" is the work of a Swiss director "Noemi Schneider" born in 1986 [19] and the director "Noemi Schneider" nominated for "DOK.fest" is from Germany, born in 1982 [18]. That is, t_s includes the information of two distinct directors, *e.g.,* the film of one and the award nomination of the other; this is called a *conflation/mismatch* in [8].

One might want to correct t_s by applying a data repairing method. It may correct errors in attributes nationality and film for the German director, but *drop the information* of the Swiss one, or the other way around. In practice, however, we want to preserve the information of both directors, without loss of information. As evidenced by IMDb and Wikidata, tuple splitting is needed instead.

Ideally, we want to split t_s into t_a and t_b for the Swiss and the German, respectively. This is, however, nontrivial. Should we assign festCity = "Munich" to t_a or t_b? What values should we assign to these attributes in the other tuple, which may become null? How can we correct the error born = "2013" in the same process? Can we complete the tuples by filling in the missing values, (*e.g.,* college)?

As another example, consider tuple t_c in Fig. 1; there is an erroneous value in nationality of t_c since no Japanese named "Noemi Schneider" was known in the film industry [16] and all other values of t_c match the information of the Swiss director [19]. In contrast to t_s, we should correct this error for the Swiss director, not to split t_c into two. The question is how to decide when we should split a tuple (*e.g.,* t_s) and when to correct errors without splitting (*e.g.,* t_c)? □

TS is as difficult as ER, if not harder. Several open issues need to be settled, which are not encountered when we conduct ER, as indicated in Example 1. How can we decide whether a tuple with conflict values should be split or corrected? To split a tuple, how should we distribute its attribute values to the right entities? How can we fill in missing values for the tuples resulted from splitting?

Contributions & organization. This paper makes a first attempt to systematically study the problem of tuple splitting (TS).

(1) A scheme (Section 2). We formulate the tuple splitting problem. Taking reliable knowledge graphs as an input, we propose a scheme, SET (Splitting EnTities), to split mismatched tuples and correct tuples. SET decides what tuples to split, splits them into multiple ones, and imputes missing values of split tuples, when tuples were mistakenly merged, the ER method for merging is unknown, and the original data is no longer there [8]. It corrects errors with certainty (see below), no matter whether the tuples are split or not.

(2) Extending REEs (Section 3). We propose an extension of Entity Enhancing Rules (REEs) of [51, 55, 57], refereed to as REE$^+$s. REE$^+$s support (a) ML models for determining correlated values as predicates, (b) predicates for heterogeneous entity resolution (HER) [50] across relations and knowledge graphs G, and (c) predicates for imputing missing values by data extraction from G. By employing REE$^+$s, SET splits mismatched tuples and corrects errors in a uniform process of logic deduction, ML correlation and data extraction.

(3) Detecting mismatched entities (Section 4). SET decides what tuples to split and what tuples to correct, by embedding an ML model $\mathcal{M}_c$ that assesses the correlation of attribute values as a predicate in REE$^+$s. For a tuple to split, it decomposes it into multiple tuples, each denotes a distinct entity, by referencing knowledge graph G. Departing from existing models, $\mathcal{M}_c$ is trained with context-aware embeddings, based on both knowledge graphs and language models.

<u>*(4) Splitting tuples*</u> (Section 5). In a uniform process, SET splits and corrects tuples. It extends the chase [111] with a set Σ of REE$^+$s and a conflict resolution strategy. For a tuple to split, it distributes attribute values to right entities. For tuples with errors, it resolves the conflicts by enforcing REE$^+$s and accumulating/referencing a set Γ of validated facts (ground truth). We show that under certain conditions on the ML models $\mathcal{M}$ in Σ, the chase is Church-Rosser [23], *i.e.,* it converges at the same result no matter in what order the rules are applied. The fixes are *certain*, *i.e.,* they are logical consequence of Σ and Γ, and are correct as long as Σ, Γ and $\mathcal{M}$ are correct/accurate.

<u>*(5) Deducing missing values*</u> (Section 6). SET fills in the missing values of the split tuples by supporting three strategies: logic deduction, data extraction from knowledge graphs, and ML prediction. It trains an ML model $\mathcal{M}_d$ for suggesting values. It unifies the three strategies and completes the split tuple by chasing with REE$^+$s, prioritizing the first two strategies. Our method also works for imputing incomplete information in general, a topic for which effective methods remain to be developed, not limited to tuple splitting.

<u>*(6) Experimental study*</u> (Section 7). Using real-life data, we empirically verify the accuracy and efficiency of the tuple splitting scheme. We find the following. On average, (a) its F_1-score is 0.92 by combining logic deduction, ML correlation models and data extraction from knowledge graphs. It is more accurate than all the baselines, by 31.8%, 8.3% and 39.5% for deciding what tuples to split/correct, assigning attribute values to the split tuples, and imputing missing value, respectively. It outperforms rule-based methods and ML-based methods by 35.5% and 30.3% respectively. (b) It takes 1,481s on a dataset of 1,057,217 tuples, with a single machine.

Related work. We categorize the related work as follows.

<u>*Entity resolution.*</u> Prior ER methods can be classified as follows. (1) Rule-based: uniqueness-constraints [66], matching dependencies (MDs) [30, 48, 80, 112, 115], pairwise-comparison [123], similarity-comparison [109], rule learning by examples [113], blocking approaches [32, 64] and datalog-like constraints [25]. (2) ML models: deep learning (*e.g.,* [46, 58, 86, 88, 99, 135]), active learning (*e.g.,* [24, 95, 105]), and unsupervised learning (*e.g.,* [124, 131, 132]). (3) Hybrid: [27, 42] approach ER by combining ML models with logical rules, and REEs [55, 57] embed ML models as predicates.

The need for distinguishing mismatched entities has long been recognized. [48, 57, 122] specify entities that should not be matched by rules. [127] identifies hard conflicts on single attributes. [36] studies soft conflicts on multiple attributes. [67] discovers mis-classified entities from a group of categorized entities (*entity categorization*). [35, 72] study the risks of entity pairs being mismatched.

To our knowledge, no prior work has studied TS. (1) We target tuples of mismatched entities that are present in our datasets, despite the effort of preventing so. (2) As the inverse of ER, TS requires to detect, split, and complement mismatched entities, beyond the tasks of ER. (3) We propose the first approach to splitting tuples, by unifying logic, ML and data extraction from knowledge bases.

<u>*Missing value imputation.*</u> The prior work is classified as follows.

(1) Rules: Functional dependencies (FDs), conditional functional dependencies (CFDs) [49], denial constraints (DCs) [26], pattern functional dependencies (PFDs) [104] and REEs [55] could be used for imputation. [110] iteratively applies FDs; [89] estimates the possible ranges of aggregate queries; [104] uses PFDs with regex expressions; [62] adopts DCs for probabilistic repair; [116] employs differential dependency in relations; [114] recovers missing attributes and links in graphs, and [57] detects errors by extending hypercube.

(2) ML models: (a) deep learning, *e.g.,* Restore [69] on relational data, DeepMVI [28] for time series, AimNet [125] and Datawig [31] for structural mixed data; notably [65, 94, 101] adopt autoen-

coder, EDIT [97] and MIWAE [93] consider missing values in training, and IPM [96] incorporates imputation semantics into pre-trained language models; (b) generative adversarial net (GAN), *e.g.,* GAIN [128] and GINN [117] impute missing values when training data is incomplete, small or noisy, and SSGAN [98] imputes multivariate time series; (c) transfer learning, *e.g.,* Baran [91] learns models from external sources to infer missing values in similar domains; (d) other approaches, *e.g.,* IIM [130], RRSI [100], ORBITS [77] and SOFIA [82], that adopt ML techniques to impute values. Off-the-shelf imputation methods might introduce bias in datasets; this issue can be tackled by considering fairness [134], learning specific tasks [60] or a missingness graph [81] during imputation.

(3) Hybrid: [70, 102, 133] integrate rule learning from knowledge graphs (KGs) with embedding models, to infer missing triples in KGs. An evaluation of imputation for time series is in [78].

This work differs from the prior work in the following. (1) Missing value imputation is just one step of SET; the prior methods cannot be directly used to split tuples. (2) We propose a logical framework that embed logic deduction, ML models and data extraction in the same chase process, beyond statistical learning and inference [108]. (3) Besides missing values, SET also corrects errors in the same process. (4) We train an ML model to deduce missing values based on attribute correlations, beyond probabilistic inference.

Error correction. There has also been work on error correction. (1) Rule-based methods: Heuristic fixes [26, 29, 39, 45, 61, 63, 108] and certain fixes [53–56], *e.g.,* [63] uses cascade repairing to correct data with minimal changes, and LLUNATIC [61] employs chase to clean data by integrating user interaction and value confidence. In contrast, we aim to split mismatched tuples and and fix errors with certainty, as opposed to minimal changes [63]; our extended chase supports a learning-based conflict resolution strategy and has the Church-Rosser property, which is not guaranteed by [61]. (2) ML-based methods: Baran [91] adopts feature engineering to generate features and then passes them to ML models for correction. SCARE [126] combines ML models and likelihood methods for data cleaning. (3) Bayesian methods: PClean [85] and BayesWipe [41] adopt Bayesian generative models to clean data injected with prior knowledge. (4) ML pipelines: CleanML [87] and Picket [90] correct data errors in ML pipelines to improve ML models. [43] explains results of data cleaning methods based on shapley values.

This work extends error correction with tuple splitting (TS); we study TS, a new problem. While error correction aims at repairing *individual* tuples only, TS splits each tuple to multiple, one for each entity, and corrects the split tuples for *all* entities. This said, SET deduces certain fixes and imputes missing values in the same process, as logical consequences of REE$^+$s and ground truth. It employs a powerful set of rules: REEs of [55, 57] already subsume CFDs, DCs and MDs as special cases, and support entity resolution and conflict resolution; moreover, REE$^+$s extend REEs with correlation models, HER and data extraction for TS and missing value imputation.

2 SPLITTING MISMATCHED TUPLES

In this section, we first formulate the tuple splitting problem (Section 2.1). We then present a tuple splitting scheme (Section 2.2).

2.1 The Tuple Splitting Problem

We start with basic notations about relations and graphs.

Preliminaries. Consider a relation schema $R = (A_1 : \tau_1, \ldots, A_n : \tau_n)$ with attributes A_i of type τ_i ($i \in [1, n]$). A relation of R is a set of tuples $(A_1 = c_1, \ldots, A_n = c_n)$, where c_i is either a constant of type τ_i, or null (when the value of the A_i-attribute is missing). We assume *w.l.o.g.* that each tuple is identified by tid, the tuple id.

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.

Entity resolution. Following Codd [37], consider tuples of schema R that denote entities in a (countably infinite) set $\mathcal{E}$ of entities. Denote by D_e the set of tuples of R such that each tuple $t \in D_e$ represents a distinct entity in $\mathcal{E}$. Assume a bijective mapping f from D_e to $\mathcal{E}$ such that for any $t \in D_e$, $f(t)$ is the entity denoted by t.

Informally, given a relation of schema R, an entity resolution method ER is to identify tuples such that for any t_1 and t_2 in the relation, if t_1 and t_2 denote the same entity in $\mathcal{E}$, then $\mathsf{ER}(t_1, t_2) = \text{true}$. In practice, ER often identifies multiple tuples and merges them into the same tuple. Ideally, $\mathsf{ER}(t_1, t_2) = \text{true}$ iff $f(t_1) = f(t_2)$.

We consider a relation D of schema R possibly after ER is applied to it. In the real world, D may have tuples of mismatched entities. Multiple tuples are merged into the same t but they denote distinct entities, *i.e.*, $\mathsf{ER}(t_1, t_2) = \text{true}$ but $f(t_1) \neq f(t_2)$. Here t_1 and t_2 are mistakenly matched to the same t, because they may bear erroneous values and/or the ER method is not very accurate.

The tuple splitting problem. Tuple splitting aims to develop a function TS such that (1) if $f(t) \notin \mathcal{E}$ (*i.e.*, t does not refer to a unique entity in $\mathcal{E}$), TS decomposes t into a minimum set $\mathsf{TS}(t) = \{t_1, \ldots, t_k\}$ such that $f(t_i) \in \mathcal{E}$ and $f(t_i) \neq f(t_j)$ for $i \neq j$ and $i, j \leq k$, where the attribute values of t_i are either inherited from t (possibly corrected if erroneous) or deduced/predicted via correlated attribute values in t; and (2) if $f(t) \in \mathcal{E}$ but t contains conflicting values (*e.g.*, "Japanese" of t_c in Figure 1), TS corrects the errors in t without splitting. We refer to $\mathsf{TS}(t)$ as a *split of* t.

Intuitively, TS splits tuples of mismatched entities, and corrects erroneous values in all the tuples (split or not), in the same process. Here $\mathsf{TS}(t)$ denotes the correction of tuple t, consisting of split and corrected tuples of t. In particular, when $|\mathsf{TS}(t)| = 1$, $\mathsf{TS}(t)$ simply corrects the conflicting attribute values of t, without splitting t.

More formally, the *tuple splitting problem* is stated as follows.

- *Input*: A schema R, a relation D of R, and a knowledge graph G.
- *Output*: The split $\mathsf{TS}(t)$ for all $t \in D$, possibly by referencing G.

Recall tuple t_s in Example 1. $\mathsf{TS}(t_s)$ splits t_s into t_a and t_b to represent the Swiss and German director, respectively. Moreover, $\mathsf{TS}(t_s)$ corrects the errors and fills in missing values in t_a and t_b. For tuple t_c, $|\mathsf{TS}(t_c)| = 1$, and we correct its errors without splitting.

This is nontrivial since D_e and $\mathcal{E}$ are often *not* known. To compute $\mathsf{TS}(t)$, we need to decide whether to split t. To split t, we have to not only decide to which entity ($f(t_a)$ or $f(t_b)$) each attribute value $t[A]$ belongs, but also fill in missing values in t_a and t_b that are inevitable from splitting. Add to the complication that attribute values are often erroneous; we have to detect and correct the errors when computing $\mathsf{TS}(t)$, no matter whether t is to be split or not.

2.2 A Scheme for Splitting Tuples

We next present a scheme for splitting tuples of mismatched entities, referred to as SET (Splitting EnTities), which subsumes error correction. As shown in Figure 2, SET takes as input a relation D of schema R, and a reliable knowledge graph G. It works in four steps, possibly interacting with the users to confirm its decision.

(1) Identifying tuples to split (DecideTS). For each t in D, SET detects conflicts in a single tuple

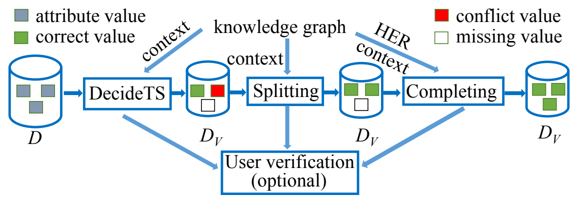


Fig. 2. The workflow of SET

(*e.g.,* a film and filmFestival) and across tuples (*e.g.,* different countries for the same city). It trains a model $\mathcal{M}_c$ offline to measure the correlation of attributes. It employs REE$^+$s (Section 3) to identify tuples with *conflicting attributes*, by taking $\mathcal{M}_c$ as predicates. It returns a set $D_V \subseteq D$ of tuples with conflicts. For each $t \in D_V$, SET creates a set $\mathsf{TS}(t)$ of split tuples $\{t_1, ..., t_k\}$ based on conflicting attributes, by consulting knowledge graphs or users, such that each t_i denotes a distinct entity. When $|\mathsf{TS}(t)| = 1$, t is erroneous and is corrected without splitting.

(2) Splitting and correcting tuples (Splitting). For each t in D_V to split or correct, SET resolves conflicts and distributes attribute values of t to the right entities $f(t_i)$ ($i \in [1, k]$) in a uniform process, by chasing $\mathsf{TS}(t)$ with REE$^+$s, which checks attribute correlation via $\mathcal{M}_c$. It corrects errors of all t in D_V, including but not limited to split tuples. The chase accumulates and references ground truth. The corrections are accurate under certain conditions (see Section 5).

(3) Deducing missing values (Completing). SET then fills in missing values of tuples in $\mathsf{TS}(t)$ by applying REE$^+$s. It trains an ML prediction model $\mathcal{M}_d$ offline to suggest missing values. Using REE$^+$s, SET uniformly deduces missing values via logic deduction, extracts data from knowledge graphs, and infers missing values with $\mathcal{M}_d$.

(4) User verification. SET presents tuples in $\mathsf{TS}(t)$ to users for confirmation. We accumulate (manually or automatically) verified values in a set Γ of ground truth, which is referenced in steps (1)-(3).

SET differs from traditional data cleaning methods in that it identifies tuples of mismatched entities and splits such tuples. As will be seen in Section 7, these improve the overall accuracy in the presence of mismatched entities. SET subsumes error correction in that it fixes errors, no matter whether the tuples are split or not.

Knowledge graphs. SET mines REE$^+$s using methods [51] and accumulates ground truth itself (with possible user verification). It takes knowledge graphs (KGs) as input, for heterogeneous ER (HER) and ML pre-training. Several popular KGs are in place, *e.g.,* Freebase [34], DBpedia [83], Yago [118] and domain-specific DRKG [74] for drugs. We can select appropriate KGs based on the application needs; the discovery/construction of KGs is beyond the scope of this paper. While (clean) KGs are not a must (by using REEs of [55, 57]), SET performs well only in some cases without KGs (see Section 7).

Limitations. Since SET takes KGs as input and employs REE$^+$s to split tuples, its effectiveness is heavily affected by the quality of REE$^+$/KGs used, which in turn depends on the underlying rule mining and KG cleaning methods. Moreover, when the training data is insufficient/unrepresentative, data imputation via correlation analysis may possibly exacerbate the problem of bias in the datasets. We defer the study of these issues as topics for future work.

We will present Steps 1-3 in Sections 4-6, respectively.

3 EXTENDING ENTITY ENHANCING RULES

In this section, we introduce REE$^+$s, an extension of entity enhancing rules (REEs) [51, 55, 57] by supporting predicates for ML correlation models and data extraction from knowledge graphs.

Below we first review the definition of REEs of [55]. We then present REE$^+$s.

Review. REEs are originally defined over a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where each R_i is a relation schema (Section 2.1).

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

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

where $\otimes$ is a comparison operator $=, \neq, <, \leq, >, \geq$. Following tuple relational calculus [23], (1) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is called a tuple variable *bounded by* $R(t)$. (2) When t is bounded by $R(t)$ and A is an attribute of R, $t[A]$ denotes the A-attribute of t. (3) In $t[A] \otimes c$, c is a constant in the domain of attribute A in R. (4) In $t[A] \otimes s[B]$, $t[A]$ and $s[B]$ are *compatible*, *i.e.,* t (resp. s) is a tuple of some relation R (resp. R'), and $A \in R$ and $B \in R'$ have the same type. Moreover, (5) $\mathcal{M}$ is an ML classifier, and $t[\bar{A}]$ and $s[\bar{B}]$ are vectors of pairwise compatible attributes of t and s, respectively.

Intuitively, $\mathcal{M}$ is an ML model that returns a Boolean value. We consider $\mathcal{M}$ such as (1) NLP models, *e.g.,* Bert [44], for text classification; (2) ER models and link prediction models, *e.g.,* Bert [44] for semantic matching; and (3) models for error detection and correction, *e.g.,* generative models [128]. We refer to $\mathcal{M}$ as an *ML predicate*.

REEs. An entity enhancing rule φ over schema $\mathcal{R}$ is defined as

$$X \rightarrow e.$$

Here (1) X is a conjunction of predicates over $\mathcal{R}$, and (2) e is a predicate over $\mathcal{R}$ such that all tuple variables in φ are bounded in X. We refer to X and e as the *precondition* and *consequence* of φ, respectively.

As shown in [55, 57], REEs subsume CFDs [49], DCs [26] and MDs [48] as special cases. In addition, REEs support ML predicates. REEs have been being employed by Rock, an industrial scale system, to catch duplicates (ER) and conflicts (CR, conflict resolution).

Semantics. Consider a database $\mathcal{D}$ of schema $\mathcal{R}$. A *valuation* of tuple variables of an REE φ in $\mathcal{D}$, or simply *a valuation of* φ, is a mapping h that maps each t in relation atom $R(t)$ of φ to a tuple in the relation of schema R in $\mathcal{D}$. We say that h *satisfies* a predicate p, written as $h \models p$, if (1) when p is $R(t)$, $t[A] \otimes c$ or $t[A] \otimes s[B]$, $h \models p$ is interpreted as in tuple relational calculus [23]. (2) When p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, $h \models p$ 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 $h \models e$. A database $\mathcal{D}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{D} \models \varphi$, if *for all* valuations h of φ in $\mathcal{D}$, $h \models \varphi$. We say that $\mathcal{D}$ *satisfies* a set Σ of REEs, denoted by $\mathcal{D} \models \Sigma$, if for all $\varphi \in \Sigma$, $\mathcal{D} \models \varphi$.

Extending REEs. We next extend REEs by supporting the following predicates defined over a database schema $\mathcal{R}$ and a knowledge graph G, in addition to the predicates given above:

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

Here (a) x in $\mathsf{vertex}(x, G)$ is a variable denoting a vertex in knowledge graph G, referred to as a *variable bounded by* $\mathsf{vertex}(x, G)$. (b) If x is bounded by $\mathsf{vertex}(x, G)$ and t is bounded by $R(t)$, $\mathsf{HER}(t, x)$ 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, $\mathsf{match}(t.A, x.\rho)$ 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 $\mathsf{match}(t.A, x.\rho)$ returns true, $t[A] = \mathsf{val}(x.\rho)$ 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) As will be seen in Section 4, $\mathcal{M}_c$ is an ML model that checks the strength of the correlation between (partial) tuple $t[\bar{A}]$ and the B-attribute value $t[B]$, and δ is a strength threshold. (f) We will see in Section 6 that $\mathcal{M}_d$ is an ML model that given a partial tuple $t[\bar{A}]$, predicts a value for its B-attribute.

We remark the following about these new predicates.

(1) SET supports the following methods for implementing $\mathsf{HER}(t, x)$ (heterogeneous entity resolution): rule-based JedAI [103], parametric simulation [50], and ML models Silk [75] and MAGNN [59].

(2) One can implement $\mathsf{match}(t.A, x.\rho)$ by using a Long-Short Term Memory (LSTM) network [71] as shown in [50].

(3) Predicates $\mathsf{vertex}(x, G)$, $\mathsf{HER}(t, x)$, $\mathsf{match}(t.A, x.\rho)$ and $t[A] = \mathsf{val}(x.\rho)$ 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) Predicates $\mathcal{M}_c(t[\bar{A}], t[B]) \otimes \delta$ and $\mathcal{M}_c(t[\bar{A}], t[B] = c) \otimes \delta$ assess correlation between values, and $t[B] = \mathcal{M}_d(t[\bar{A}], B)$ suggests a value for (missing) attribute B. We refer to them as *correlation predicates*. We will train $\mathcal{M}_c$ and $\mathcal{M}_d$ in Sections 4 and 6, respectively.

(5) We define REE$^+$s as a general extension of REEs such that existing REE applications can be extended to REE$^+$s. This said, we use REE$^+$s of special forms for different steps of tuple splitting; *e.g.*, the $\mathcal{M}_c$ model is used for deciding tuples to split/correct (Section 4) and attribute assignment (Section 5); and the $\mathcal{M}_d$ model and extraction predicates are mostly used for imputing missing values (Section 6).

REE$^+$s. REE$^+$s also have the form $\varphi = X \rightarrow e$, except the following: all tuple variables and vertex variables in φ are bounded in X.

Example 2: Below are some REE$^+$s over the schema of Example 1.

(1) $\varphi_1 = \mathsf{person(t)} \rightarrow \mathcal{M}_c(t[\mathsf{film}], t[\mathsf{filmFestival}]) \geq \delta$, where $\mathcal{M}_c$ checks the correlation between attribute values, and δ is a predefined threshold. It says that in tuple t, film and filmFestival should be strongly correlated. We will see (in Section 4) that t_s of Example 1 needs to be split/corrected since $\mathcal{M}_c$("Sturm", "DOK.fest") is small.

(2) $\varphi_2 = \mathsf{person}(t) \wedge \mathcal{M}_c(t[\mathsf{name}, \mathsf{filmFestival}], t[\mathsf{festCity}] = c_1) \geq \delta \rightarrow t[\mathsf{festCity}] = c_1$. We will see in Section 5 that φ_2 helps us distribute values to split tuples; it decides festCity = "Munich" for the German director, since DOK.fest is an annual event in Munich.

(3) $\varphi_3 = \mathsf{person}(t) \wedge t[\mathsf{festCity}] =$ "Munich" $\rightarrow t[\mathsf{festCountry}] =$ "Germany". As will be seen in Section 6, φ_3 can be used for deducing the missing value of festCountry based on the value of festCity.

(4) $\varphi_4 = \mathsf{person}(t) \wedge \mathsf{null}(t[\mathsf{college}]) \rightarrow t[\mathsf{college}] = \mathcal{M}_d(t[\mathsf{name}, \mathsf{film}], \mathsf{college})$, where $\mathsf{null}(t[A])$ is a syntactic abbreviation to check whether $t[A]$ carries null value (*i.e.,* $t[A]$ "=" null), and ML model $\mathcal{M}_d$ predicts missing values. Intuitively, we can use φ_4 to find college = "ZHdk" for the Swiss director, since "Sturm" is the degree film of the Swiss director during her bachelor study at "ZhdK" [3] (as evidenced in [11, 15], ZhdK produces films for its students).

(5) $\varphi_5 = \mathsf{person(t)} \wedge \mathsf{vertex}(x, \mathrm{Wiki}) \wedge \mathsf{HER}(t, x) \wedge \mathsf{match}(t[\mathsf{born}], x.(\mathrm{yearOfBirth})) \rightarrow t[\mathsf{born}] = \mathsf{val}(x.\mathrm{yearOfBirth})$. This REE$^+$ says that if a person t in D matches a person vertex x in Wiki and if x reaches vertex v via an one-hop path ρ = (yearOfBirth), then let $t[\mathsf{born}]$ take $L(v)$ as its value. As will be seen in Section 6, this is how we correct the erroneous value $t_s[\mathsf{born}]$ = "2013" and fetch the correct year of birth for both directors. Similarly, we can extract data from Wiki and (optionally) impute other null values for them.

(6) $\varphi_6 = \mathsf{person}(t) \wedge \mathsf{person}(s) \wedge t[\mathsf{college}] = s[\mathsf{college}] \rightarrow t[\mathsf{Country}] = s[\mathsf{Country}]$, assuming the existence of attribute Country (not shown). It states a regularity that the same college must be in the same country, used for (a) detecting conflicts, as evidences of split/corrections, and (b) correcting errors on Country. □

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.

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 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 $\mathcal{M}_c(t[\bar{A}], t[B] = c) \otimes \delta$ (resp. $\mathcal{M}_c(t[\bar{A}], t[B]) \otimes \delta$), let d be the strength of the correlation between $h(t)[\bar{A}]$ and c (resp. $t[B]$) assessed by $\mathcal{M}_c$, then $d \otimes \delta$. (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 $\mathcal{M}_d$ to extend partial tuple $t[\bar{A}]$.

Discovery of REE$^+$s. Algorithms are in place for discovering REEs of [55, 57], *e.g.,* [51, 52]. We extend them to discover REE$^+$s as follows. We adopt levelwise search to mine REE$^+$s $\varphi : X \rightarrow e$, where X is empty initially. We iteratively pick a predicate p and extend X to $X \wedge p$ until (a) there is no predicate to be selected, or (b) $\varphi : X \rightarrow e$ is qualified to be returned (*e.g.,* its confidence is above a threshold).

4 DECIDING TUPLES TO SPLIT/CORRECT

In this section, we first train an ML model $\mathcal{M}_c$ for assessing the correlation between attribute values. We then present our method for deciding what tuples to split and what tuples to correct by embedding $\mathcal{M}_c$ in REE$^+$s, and consulting knowledge graphs/users.

4.1 Correlation Model $\mathcal{M}_c$

The correlation 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 a confidence (in $[0, 1]$) indicating the strength of correlation between $t[\bar{A}]$ and $t[B]$. Intuitively, the higher the correlation strength is, the more likely $t[\bar{A}]$ and $t[B]$ coexist in an entity; a small strength means that t might contain conflicts and thus, need to be split or corrected.

Challenges. One may want to adopt an existing model (*e.g.,* LSTM) to learn a representation of $(t[\bar{A}], t[B])$. But it does not work well.

(1) Prior knowledge. To determine the correlation between $t[\bar{A}]$ and $t[B]$, one often has to reference other sources for additional hints, *e.g.,* if we know "DOK.fest is an annual event held in Munich", then $t_s[\bar{A}]$ ="Munich" and $t_s[B]$ ="DOK.fest" are likely to be correlated.

(2) Limitation of embedding models. Pre-trained embedding models are mostly trained on unstructured text, rather than structured relational data. Moreover, they are not purposely trained to assess the correlation between different attribute values. To utilize the embedding models, we need a mechanism to bridge the gap.

(3) Training data. To learn correlation, a large amount of training instances is a must. It is unrealistic to label them by hand. Worse still, it requires strong background, *e.g.,* only film fans may know that DOK.fest is held annually in Munich, and label them as correlated.

Model. To tackle these challenges, we propose a model $\mathcal{M}_c$, whose novelty includes (a) a pretraining procedure to incorporate prior knowledge into $(t[\bar{A}], t[B])$ based on *knowledge graphs*, (b) a well-designed encoding scheme so that we can predict correlation by utilizing existing embedding techniques, and (c) a self-supervised learning mechanism to train $\mathcal{M}_c$ without heavy human labeling.

As shown in Figure 3, $\mathcal{M}_c$ takes $t[\bar{A}]$ and $t[B]$ as input, and outputs a confidence indicating their correlation. It has two major steps.

(1) Graph pretraining. We pretrain graph embeddings on a knowledge graph G, so that we can implicitly learn rich contextual information (*e.g.,* DOK.fest held in Munich) from pretrained embedding.

(2) Context-aware embedding. We model $I_t = (t[\bar{A}], t[B])$ as a sequence (by concatenating attribute

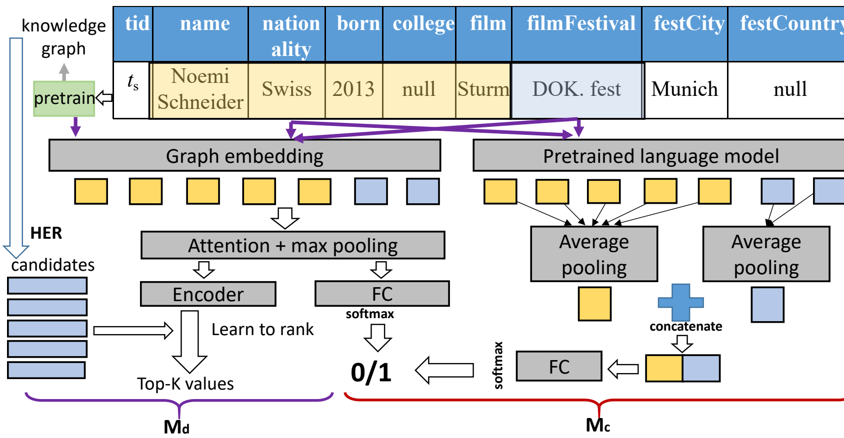


Fig. 3. The Network Architecture of $\mathcal{M}_c$ and $\mathcal{M}_d$

values) and design encoders to obtain two representations of I_t, via *graph embeddings* and *language models*, respectively. After a softmax layer, we combine the classifications and generate a confidence score by incorporating semantics.

Graph pretraining. Given a knowledge graph $G(V, E, L)$, we learn node/relation embeddings using graph representation methods [84], such that given a score function $g(\cdot)$ on node embeddings $\mathbf{u}, \mathbf{v}, \mathbf{v}'$ of $u, v, v' \in V$ and relation embedding $\mathbf{r}$ of edge label r, where $e = (u, v)$ with $L(e) = r$ is in E and $e' = (u, v')$ with $L(e') = r$ is not in E, $g(\mathbf{u}, \mathbf{r}, \mathbf{v})$ is maximized, while $g(\mathbf{u}, \mathbf{r}, \mathbf{v}')$ is minimized. This said, we learn the embeddings such that if the embeddings of two vertices are close, they are correlated. Graph pretraining is executed once as a preprocessing step and will not increase the cost of inference.

Context-aware embedding. We treat $I_t = (t[\bar{A}], t[B])$ as a sequence, by concatenating attribute values $t[\bar{A}]$ and $t[B]$. To get the representation for I_t, we adopt two embedding mechanisms.

(a) Graph embeddings. We tokenize I_t and obtain the token embeddings based on a lookup table Dict from graph pretraining on different knowledge graphs (*e.g.*, [21]). Specifically, if a token T is found in Dict, we embed it as $\mathbf{T} = \mathsf{Dict}[T]$; otherwise, we randomly initialize its embedding with the normal (Gaussian) distribution. Then we transform I_t into a matrix $\mathbf{M}_{\mathrm{G}} = [\mathbf{T}_1; \ldots; \mathbf{T}_{|I_t|}] \in \mathbb{R}^{|I_t| \times d_1}$, where d_1 is the dimension of graph embeddings. We encode I_t based on $\mathbf{M}_{\mathrm{G}}$ as follows, denoted by $\mathsf{h}(\mathbf{M}_{\mathrm{G}}) \in \mathbb{R}^{d_1 \times 1}$:

$$\mathsf{h}(\mathbf{M}_{\mathrm{G}}) = \mathsf{Encoder}_{\mathrm{G}}(\mathbf{M}_{\mathrm{G}}) = \mathsf{Pool}_{\mathsf{max}}(\mathsf{Attention}(\mathbf{M}_{\mathrm{G}})),$$

where $\mathsf{Encoder}_{\mathrm{G}}$ is the encoder of graph embedding with the attention mechanism [121] Attention and max pooling strategy $\mathsf{Pool}_{\mathsf{max}}$.

(b) Language models. To create representations based on language models, we adopt serialization [88]. Specifically, we serialize I_t:

$$\mathsf{serial}(I_t) = \langle\mathsf{COL}\rangle A_1 \langle\mathsf{VAL}\rangle t[A_1] \ldots \langle\mathsf{COL}\rangle A_k \langle\mathsf{VAL}\rangle t[A_k] \langle\mathsf{COL}\rangle B \langle\mathsf{VAL}\rangle t[B],$$

where $\bar{A} = \{A_1, \ldots, A_k\}$, $\langle\mathsf{COL}\rangle$ and $\langle\mathsf{VAL}\rangle$ are special tokens [88] indicating the start of attribute and value, respectively. The serialization of I_t is fed to a language model LM (*e.g.*, [44]). Given a token T in $\mathsf{serial}(I_t)$, we use $\mathbf{T} = \mathsf{LM}(T)$ as its embedding. Then, we transform I_t into a matrix $\mathbf{M}_{\mathrm{LM}} = [\mathbf{T}_1; \ldots; \mathbf{T}_{|\mathsf{serial}(I_t)|}] \in \mathbb{R}^{|\mathsf{serial}(I_t)| \times d_2}$, where d_2 is the dimension of language model embedding. Similarly, we encode a representation $\mathsf{h}(\mathbf{M}_{\mathrm{LM}}) \in \mathbb{R}^{d_2 \times 2}$ as follows:

$$\mathsf{h}(\mathbf{M}_{\mathrm{LM}}) = \mathsf{Encoder}_{\mathrm{LM}}(\mathbf{M}_{\mathrm{LM}}) = [\mathsf{Pool}_{\mathsf{avg}}([\mathbf{T}_{\bar{A}}]); \mathsf{Pool}_{\mathsf{avg}}(\mathbf{T}_B)],$$

where $\mathbf{M}_{\mathrm{LM}}$ is written as $[\mathbf{T}_{\bar{A}}, \mathbf{T}_B]$ with the embedding matrices of $t[\bar{A}]$ and $t[B]$, respectively, and Pool_{avg} is the average pooling.

Confidence. Finally, we generate the confidence of the correlation between $t[\bar{A}]$ and $t[B]$, by applying a fully-connected layer (FC) and softmax activation to compute 2-dimensional probabilities:

$$\mathbf{p}_{\mathsf{G}} = \mathsf{Softmax}(\mathsf{FC}_{\mathsf{G}}(\mathsf{h}(\mathbf{M}_{\mathsf{G}}))), \quad \mathbf{p}_{\mathsf{LM}} = \mathsf{Softmax}(\mathsf{FC}_{\mathsf{LM}}(\mathsf{h}(\mathbf{M}_{\mathsf{LM}}))),$$

where $\mathbf{p}_{\mathsf{G}}[0]$ (resp. $\mathbf{p}_{\mathsf{G}}[1]$) is the probability that $t[\bar{A}]$ and $t[B]$ are (resp. not) correlated based on the graph embeddings; similarly for $\mathbf{p}_{\mathsf{LM}}$ which is obtained based on the language model embeddings.

Then the final confidence value for $t[\bar{A}]$ and $t[B]$ is:

$$\mathcal{M}_c(t[\bar{A}], t[B]) = \alpha \cdot \mathbf{p}_{\mathsf{G}}[0] + (1-\alpha) \cdot \mathbf{p}_{\mathsf{LM}}[0],$$

where α is a hyper-parameter to balance the two mechanisms. Intuitively, in this way, we not only utilize contextual knowledge from knowledge graphs, but also augment the result with rich semantics.

Loss function and training strategy. Let $\mathcal{T}_c = \{(x_i, y_i)\}_{i=1}^N$ be the set of training data, where $x_i = (t_i[\bar{A}], t_i[B])$ is the i-th training data and $y_i \in \{0, 1\}$ is its label; $y_i = 1$ if $t_i[\bar{A}]$ and $t_i[B]$ are correlated and $y_i = 0$ otherwise. We adopt the cross entropy loss as follows:

$$\mathcal{L}_{\mathsf{CE}}(\mathcal{T}_c) = -\frac{1}{N} \sum_{(x_i, y_i) \in \mathcal{T}_c} (y_i \cdot \log(\mathcal{M}_c(x_i)) + (1 - y_i) \cdot \log(1 - \mathcal{M}_c(x_i))).$$

It is labor-intensive and needs strong background to label data. Nonetheless, for $t \in D$ and $A \in R$, $t[R_{-A}]$ is often correlated with $t[A]$ in practice, where R_{-A} is all attributes excluding A. Hence we adopt self-supervised learning and generate $\mathcal{T}_c$ as follows. (a) We randomly sample a tuple t from D. (b) We randomly select one attribute B and let the remaining be $\bar{A}$. (c) With $p = \frac{1}{2}$ probability, we use $(t[\bar{A}], t[B], 1)$ as a positive example. With $1 - p$ probability, we randomly select a value c from the domain of B and use $(t[\bar{A}], c, 0)$ as a negative example. In this way we get N examples in $\mathcal{T}_c$.

4.2 Identifying Tuples to Split and Correct

We present our method for identifying tuples of mismatched entities to split and tuples with conflicts to correct, by combining logic deduction, correlation analysis and heterogeneous ER (HER) across relations and knowledge graphs. Intuitively, t needs to be split if (1) t has conflicting values that *violate* data regularity, enforced by a set Σ_d of REE^+s on accumulated ground truth Γ; and (2) those values belong to multiple entities, confirmed by users and/or knowledge graph G. If only (1) holds, we correct errors in t without splitting.

For instance, we split t_s in Example 1, since (a) no short film "Sturm" was ever nominated at "DOK.fest" [13] (a conflict) and more importantly, (b) these two values come from the Swiss and German director, respectively [18, 19]; however, for t_c, we only correct its erroneous nationality, since "Japanese" is loosely correlated to all other values, which matches the Swiss director [19].

To detect conflicting values, we use two types of REE^+s: (a) REEs $X \to e$ of [55, 57] with predicates $R(t), t[A] \otimes c, t[A] \otimes s[B]$ and $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, and at most two tuple variables, to identify critical conflicts; and (b) REE^+s of the form $R(t) \wedge \mathcal{M}_c(t[\bar{A}], t[B]) \le \delta \to \text{false}$, where false is a syntactic sugar expressed as, *e.g.,* $t[A] \ne t[A]$; intuitively, such REE^+s catch a violation if $t[\bar{A}]$ and $t[B]$ are loosely correlated (checked by $\mathcal{M}_c$). The two types of REE^+s apply logic deduction and correlation analysis to detect conflicts, respectively.

The method references accumulated ground truth Γ, which consists of $(t[A], d)$ pairs, denoting that $t[A]$ has been validated to be d by users or by referencing knowledge graphs (KGs). In particular, numerical values are tacked by comparison predicates in REE^+s, and $\mathcal{M}_c$ treats them as text (a common practice in NLP).

Violation. Below we first formalize the notion of violations.

Given an REE$^+$ $\varphi : X \to e$ in Σ_d and a tuple t^* in D, a *violation* of φ *pertaining to* t^* is a valuation h of φ that satisfies the following:

(1) The REE$^+$ φ *pertains to* t^*, *i.e.,* if φ is an REE defined in [55, 57] (reviewed in Section 3), then one of the tuple variable in e is instantiated by t^*, *e.g.,* if e is $t[A] \otimes c$, then $h(t) = t^*$; if φ is $R(t) \wedge \mathcal{M}_c(t[\bar{A}], t[B]) \leq \delta \to \mathsf{false}$, then $h(t) = t^*$.

(2) All predicates p in X are *validated.* (a) If p is $t[A] \otimes c$, then $h(t)[A]$ is validated to be d in Γ and $d \otimes c$; similarly for $t[A] \otimes s[B]$. (b) If p is $\mathcal{M}(t[\bar{A}], s[\bar{B}])$, then for each attribute A in $\bar{A}$ (resp. each B in $\bar{B}$), $h(t)[A]$ (resp. $h(s)[B]$) is validated in Γ and $\mathcal{M}(h(t)[\bar{A}], h(s)[\bar{B}]) = \mathsf{true}$. (c) If p is $\mathcal{M}_c(t[\bar{A}], t[B]) \leq \delta$, then for each attribute A in $\bar{A}$, $h(t)[A]$ is validated in Γ and $\mathcal{M}_c(h(t)[\bar{A}], h(t)[B]) \leq \delta$.

(3) Consequence e is *violated.* Take equality as an example. (a) If e is $t[A] = c$, then $h(t)[A] \neq c$. (b) If e is $t[A] = s[B]$ and assume *w.l.o.g.* that $h(t) = t^*$, then $h(s)[B]$ is validated in Γ and $t^*[A] \neq h(s)[B]$. (c) If e is false in $R(t) \wedge \mathcal{M}_c(t[\bar{A}], t[B]) \leq \delta \to \mathsf{false}$, then $\mathcal{M}_c(h(t)[\bar{A}], h(t)[B]) \leq \delta$. In cases (a) and (b), A is the *conflicting attribute* of h; in (c), B in $\mathcal{M}_c(t[\bar{A}], t[B])$ is the *conflicting attribute.*

For a set Σ_d of REE$^+$s, we denote by $\mathsf{Vio}(\Sigma_d, \Gamma, t^*)$ the set of all violations of the REE$^+$s in Σ_d pertaining to t^*, *i.e.,* $h \in \mathsf{Vio}(\Sigma_d, \Gamma, t^*)$ if h violates at least one REE$^+$ in Σ_d pertaining to t^*.

Note that SET is able to capture violations of CFDs [49], DCs [26] and MDs [48], since REE$^+$s subsume them as special cases, *e.g.,* we can rewrite φ_3 in Example 2 as a DC: $\forall t \in \mathsf{person} : \neg(t[\mathsf{festCity}] =$ "Munich" $\wedge t[\mathsf{festCountry}] \neq$ "Germany"); it catches DC violations (*e.g.,* the festival city is "Munich" but its country is not "Germany").

Algorithm. Our method, DecideTS, takes a set D of tuples, a set Σ_d of REE$^+$s, a set Γ of ground truth and a reliable knowledge graph G as input; it decides whether tuples in D need to be split or corrected. For each tuple $t \in D$, it first computes the violations $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ by using the detection algorithm of [57]. If $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ is empty, DecideTS returns false. Otherwise, we consult users or knowledge graph G to check whether $f(t) \in \mathcal{E}$ (*i.e.,* t refers to a unique entity in $\mathcal{E}$). For instance, if tuple t maps to multiple entities in knowledge graph G by HER [50], then we know that $f(t) \notin \mathcal{E}$.

- If $f(t) \in \mathcal{E}$, t is a tuple with conflicts to correct. DecideTS simply returns $\mathsf{TS}(t) = \{t\}$ (with only non-conflicting values validated) and later corrects the conflicts via the chase (Section 5).
- If $f(t) \notin \mathcal{E}$, t is a mismatch to split. We return an initial split $\mathsf{TS}(t)$ of t (see below). Denote by D_T the set of all tuples to split.

We use D_V to denote the set of tuples t in D for which $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ is nonempty, *i.e.,* all the tuples to be split or corrected.

Initial splitting. For t in D_T, let $\mathsf{Vio}(\Sigma_d, \Gamma, t) = \{h_1, \ldots, h_l\}$, and A_h be the conflicting attribute of h. We compute $\mathsf{TS}(t)$ of t as follows: (a) $\mathsf{TS}(t)$ is initialized to $\{t'\}$, where $t'[A] = t[A]$ if $t[A]$ is validated and $t'[A] = \mathsf{null}$ otherwise; and (b) for each h in $\mathsf{Vio}(\Sigma_d, \Gamma, t)$, we create a new tuple t' in $\mathsf{TS}(t)$, such that for each attribute A, $t'[A] = t[A]$ if $A = A_h$ (we set $t'[A]$ as validated) and $t'[A] = \mathsf{null}$ otherwise. Moreover, the quasi-identifier of t (*i.e.,* attributes combined as a unique identifier [119]) is replicated at each t' in $\mathsf{TS}(t)$, *e.g.,*"Noemi Schneider" is replicated at each tuple in $\mathsf{TS}(t_s)$ (Example 3). Each t' is confirmed a distinct entity by knowledge graphs (via HER) or users. The values of t' are either inherited from t (Section 5) or deduced/predicted via correlated values (Section 6). Note that DecideTS can detect tuples merged from multiple entities, *e.g.,* when $|\mathsf{TS}(t)| > 2$, it indicates that there are multiple violations of REE$^+$s pertaining to t and we may need to split t into more than two tuples.

Example 3: Continuing Example 1, assume that $D_T = \{t\}$, $\Sigma_d = \{\varphi_7\}$, where φ_7 is $\mathsf{person}(t) \wedge \mathcal{M}_c(t[\mathsf{filmFestival}], t[\mathsf{film}]) \leq 0.8 \to \mathsf{false}$ and $\Gamma = \{(t[\mathsf{filmFestival}],$ "DOK.fest"$)\}$. Consider a valuation h_7: $t_s \mapsto t$. If $\mathcal{M}_c($"DOK.fest", "Sturm"$) = 0.1$, one can verify that h_7 is a violation of φ_7 pertaining to t_s since $t_s[\mathsf{filmFestival}]$ is validated in Γ and $\mathcal{M}_c(t_s[\mathsf{filmFestival}], t_s[\mathsf{film}])$

= $0.1 \leq 0.8$. Then $\mathsf{Vio}(\Sigma_d, \Gamma, t_s) = \{h_7\}$ and t_s is a tuple to be split or corrected; film is its conflicting attribute. Note that if $t_s[\mathsf{film}]$ is validated in Γ, filmFestival is the conflicting attribute. Suppose that we reference IMDb and confirm $f(t_s) \notin \mathcal{E}$ (*e.g.*, $f(t_s)$ fuzzily maps to both directors). Then t_s is a mismatch to be split. We get an initial split $\mathsf{TS}(t_s) = \{t_a, t_b\}$, where t_a = ("Noemi Schneider", null, null, null, "Sturm", null, null, null) and t_b = ("Noemi Schneider", null, null, null, null, "DOK.fest", null, null), with non-null values validated in Γ. □

Augmenting Γ. Initially, Γ might only contain a limited number of ground truth labeled by the user (if any). As a by-product of DecideTS, we augment Γ with additional ground truth, as follows:

(1) Given a valuation h of $\varphi : X \rightarrow e$ in Σ_d where all predicates in X are validated in Γ, we process its consequence e as follows: (a) If e is $t[A] = c$ and $h(t)[A]$ is not yet validated in Γ, we add $(h(t)[A], c)$ to Γ. (b) If e is $t[A] = s[B]$ and $h(t)[A]$ (resp. $h(s)[B]$) is not yet validated (resp. is validated to be d) in Γ, we add $(h(t)[A], d)$ to Γ.

(2) Even if $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ is empty, it does not necessarily mean that t is validated (due to the lack of REE$^+$s or Γ). Thus if $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ is empty, we optionally invite a user to confirm whether t is a tuple with conflicts to split or correct, and add non-conflicting values to Γ.

Complexity. For each t in D and each φ in Σ_d, it takes $O(|D|)$ time to enumerate valuations of φ since φ has at most two variables. Assume that the unit cost for validating a valuation is c_{valid}. Then computing $\mathsf{Vio}(\Sigma_d, \Gamma, t)$ takes $O(c_{\mathsf{valid}}|D|^2|\Sigma_d|)$ time. For each tuple t in D (resp. each tuple t' in $\mathsf{TS}(t)$ where $t \in D_T$), if we reference knowledge graphs for deciding whether $f(t) \in \mathcal{E}$ (resp. $f(t') \in \mathcal{E}$), it takes $O((|V| + |E|)^2)$ time to perform HER [50]. The process takes $O(c_{\mathsf{valid}}|D|^2|\Sigma_d| + (|D| + \sum_{t \in D_T} \mathsf{TS}(t))(|V| + |E|)^2)$ time in total.

5 SPLITTING AND CORRECTING TUPLES

In this section, we present algorithm Splitting, to correct errors and assign values of t to tuples in $\mathsf{TS}(t)$, by *chasing with* REE$^+$s. We first present the workflow (Section 5.1). We then review *the chase* [111] (Section 5.2), based on which we develop Splitting (Section 5.3).

5.1 Overall Workflow

Given datasets D_V and D, a set Σ_a of REE$^+$s and a set Γ of ground truth as input, Splitting returns a correction $\mathsf{TS}(t)$ for each tuple $t \in D_V$, where each $\mathsf{TS}(t)$ corrects erroneous values in t and assigns the values of $t[A]$ to split tuples in $\mathsf{TS}(t)$ (if t is split) in a uniform process. The fixes generated are certain [54, 56], *i.e.*, they are guaranteed to be correct, as long as the REE$^+$s and ground truth are correct, when the ML models embedded in the REE$^+$s of Σ_a are accurate.

A uniform process. Enforcing a set Σ_a of designated types of REE$^+$s (see below), we extend the chase [111] by referencing Γ, to split and correct $\mathsf{TS}(t)$ in the same process. Here an REE$^+$ in Σ_a can only be applied if its precondition is validated (see Section 4.2).

Assigning values. For a tuple t to split (*i.e.*, $t \in D_T$), we assign values of t to the right entities in $\mathsf{TS}(t)$, by embedding $\mathcal{M}_c$ in REE$^+$s.

We use two types of REE$^+$s for deciding whether the B-attribute value of $t' \in \mathsf{TS}(t)$ should be inherited from t: (a) REEs of [55, 57] with at most two tuple variables that have consequence $e : t'[B] = t[B]$; and (b) REE$^+$s of the form $R(t') \wedge \mathcal{M}_c(t'[\bar{A}], t'[B] = t[B]) \geq \delta \rightarrow t'[B] = t[B]$; intuitively, $t'[\bar{A}]$ and the value $t[B]$ are strongly correlated (checked by $\mathcal{M}_c$), and thus, we assign $t[B]$ to $t'[B]$.

Correcting errors. For each tuple t in D_V, no matter whether t is to split or not, we correct erroneous values in all tuples $t' \in \mathsf{TS}(t)$ by including the entire class of REEs of [55, 57] (Section 3) in Σ_a. As shown in [57], the REEs subsume CFDs, MDs and DCs as special cases, and are able to catch errors commonly found in practice.

5.2 Chasing with REE$^+$s and Correlation Model

We next present the chase, starting with fixes.

Fixes. We assign values and correct errors for tuples t' in $\mathsf{TS}(t)$ by deducing fixes. We maintain all the *fixes* in a set $\overline{U}$, which consists of $(t'[B], c)$ pairs, indicating that $t'[B] = c$ is deduced (here c can be $t[B]$ or a confirmed constant). Fixes are logical consequences of (Σ_a, Γ), *i.e.,* as long as Σ_a and Γ are correct and the ML models embedded in Σ_a are accurate (*e.g.,* the model $\mathcal{M}_c$), so are the fixes.

Validity. We say that $\overline{U}$ is *valid* if no $(t'[B], c_1)$ and $(t'[B], c_2)$ are both in $\overline{U}$ at the same time such that $c_1 \neq c_2$ for constants c_1 and c_2.

The chase. We deduce fixes by chasing $\mathsf{TS}(t)$ with REE$^+$s in Σ_a and ground truth in Γ. Specifically, a chase step of $\mathsf{TS}(t)$ by Σ_a at $\overline{U}$ is

$$\overline{U} \Rightarrow_{(\varphi,h)} \overline{U}',$$

where $\varphi : X \to e$ is an REE$^+$ in Σ_a and h is a valuation of φ such that (a) all predicates in X are *validated* by $\overline{U}$ (*i.e.,* the corresponding pair is in $\overline{U}$), and (b) the consequence $e : t'[B] = c$ extends $\overline{U}$ to $\overline{U}'$, by adding the pair $(t'[B], c)$ to $\overline{U}$. Here h involves at least one tuple in $\mathsf{TS}(t)$ and may reference (map variables to) other tuples in D.

Chasing. A *chasing sequence* ξ of $\mathsf{TS}(t)$ by (Σ_a, Γ) is a sequence

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

where $\overline{U}_0$ is Γ and for each $i \in [1, n]$, there exist φ in Σ_a and h of φ such that $\overline{U}_{i-1} \Rightarrow_{(\varphi,h)} \overline{U}_i$ is a valid chase step, *i.e.,* $\overline{U}_i$ is valid.

The sequence ξ *terminates* in one of the following two cases:

(a) No REE$^+$s in Σ_a can be further applied; in this case, we say that the chasing sequence ξ is valid, with $\overline{U}_n$ as its result.
(b) There exist φ, h and $\overline{U}_{n+1}$ such that $\overline{U}_n \Rightarrow_{(\varphi,h)} \overline{U}_{n+1}$ but $\overline{U}_{n+1}$ is invalid. Such ξ is invalid, with its result $\perp$ (undefined).

Example 4: Continuing with Example 3, assume that the initial split is $\mathsf{TS}(t_s) = \{t_a, t_b\}$ and $\Sigma_a = \{\varphi_2, \varphi_3\}$ from Example 2. Then we have the following chase steps of $\mathsf{TS}(t_s)$ by (Σ_a, Γ):

(1) $\overline{U}_0 \Rightarrow_{(\varphi_2,h_2)} \overline{U}_1$, where h_2 maps t of φ_2 to t_b; $\overline{U}_1$ extends $\overline{U}_0$ with $(t_b[\mathsf{festCity}],$"Munich") *i.e.,* we deduce "Munich" for t_b.

(2) The chase then deduce $t_b[\mathsf{festCountry}] =$ "Germany" by (φ_3, h_3).

This chasing sequence is valid, since each chase step is valid and moreover, no more REE$^+$s in Σ_a can be further applied. □

Church-Rosser. Following [111], we say that the chase is *Church-Rosser* if for any set Σ_a of REE$^+$s, ground truth Γ, and sets D and $\mathsf{TS}(t)$ of tuples, all chasing sequences of $\mathsf{TS}(t)$ by (Σ_a, Γ) terminate and converge at the same result, denoted by $\mathsf{Chase}(\mathsf{TS}(t), \Sigma_a, \Gamma, D)$, no matter what REE$^+$s in Σ_a are used and how they are applied.

Corollary 1: *Chasing with* REE$^+$s *(having $\mathcal{M}_c$) is Church-Rosser.* □

Proof sketch: Chasing with REEs is proven to be Church-Rosser in [55]. One can verify that the proof remains intact for REE$^+$s that are extended with the correlation model $\mathcal{M}_c$ as predicates [12]. □

5.3 Splitting and Correcting with the Chase

Although conceptually simple, we cannot split tuples by directly applying the chase, for the following reasons. (a) The enumeration of valuations is costly; moreover, a valuation h of $\varphi : X \to e$ may rely on the application of other valuations h' in previous chase steps, *e.g.,* not-yet-validated

Algorithm Splitting
Input: Dataset D, split $\mathsf{TS}(t)$ for $t{\in}D_V$, REE$^+$s Σ_a, and ground truth Γ.
Output: Updated $\mathsf{TS}(t)$ for all $t \in D_V$ with more values assigned/corrected.
1. $Q := \emptyset$; $\mathcal{H} := \emptyset$; $\overline{U} := \Gamma$; $\mathcal{S} := \emptyset$;
2. $(Q, \mathcal{S}) :=$ GenerateValuation$(D, \cup_{t \in D_V} \mathsf{TS}(t), \Sigma_a, \Gamma)$;
3. **while** $Q \neq \emptyset$ **do**
4. $\quad h := Q$.pop() where h is a valuation of $\varphi : X \rightarrow t'[B] = c$;
5. $\quad \mathcal{H} := \mathcal{H} \cup \{h\}$; $\overline{U} := \overline{U} \cup \{(t'[B], c)\}$;
6. $\quad$ **if** $\overline{U}$ is invalid, *i.e.,* $\{(t'[B], c_1), (t'[B], c_2)\} \subseteq \overline{U}$, but $c_1 \neq c_2$ **then**
7. $\qquad (t'[B], c) :=$ ResolveConflict (t', B); $\Gamma{:=}\Gamma \cup \{(t'[B], c)\}$;
8. $\qquad$ Update $\overline{U}$ and affected valuations in Q and $\mathcal{H}$;
9. $\quad$ **else** Generate new valuations to Q based on $(t'[B], c)$;
10. $\Gamma := \overline{U}$;
11. **return** $\cup_{t \in D_V} \mathsf{TS}(t)$;

Fig. 4. Algorithm Splitting

p in X may become validated after applying h'. (b) The chase may terminate at an invalid result (*i.e.,* $\perp$). If so, we need to resolve conflicts $(t'[B], c_1)$ and $(t'[B], c_2)$ for $c_1 \neq c_2$.

Novelty. To overcome these, we develop Splitting, to assign values and correct errors in $\mathsf{TS}(t)$ via the chase, with the following novelty:

(a) We maintain structures to record temporary chasing results, so that valuations can be enumerated/re-used efficiently and only affected/unchecked valuations need to be examined.
(b) We develop a learning-based conflict resolution strategy, complementing the logic deduction to decide critical values.
(c) We assign values and correct errors in the same chase process.

Algorithm. We outline Splitting in Figure 4. For each $\mathsf{TS}(t)$ of $t{\in}D_V$, it corrects errors and distributes values of t to tuples in $\mathsf{TS}(t)$.

We maintain the following in Splitting (lines 1-2): (a) A set $\mathcal{H}$ (resp. Q) of valuations that have been applied (resp. to be applied later); intuitively, they avoid the same valuation to be processed repeatedly. (b) An index $\mathcal{S}$ such that for each t' in $\mathsf{TS}(t)$ and each attribute B, $\mathcal{S}[t'[B]]$ stores the valuation h of $\varphi : X \rightarrow e$, where $t'[B]$ is in X; intuitively, when $t'[B]$ is validated, we check only valuations in $\mathcal{S}[t'[B]]$ to see whether they can be applied in subsequent steps. (c) The set $\overline{U}$ of fixes deduced, initialized to be Γ. Initially, $\mathcal{H}$ is empty; Q and $\mathcal{S}$ are initialized by generating valuations h *pertaining to* Γ (line 2), *i.e.,* at least one predicate in the precondition of h is validated by Γ, instead of generating all valuations at once.

After initialization, Splitting deduces fixes by applying valuations h of $\varphi : X \rightarrow t'[B] = c$ in Q one by one (lines 3-8). Once being applied, h is moved to $\mathcal{H}$ and $\overline{U}$ is extended with $(t'[B], c)$ (Line 5). Then we check the validity of $\overline{U}$: (1) If $\overline{U}$ is valid (line 9), we generate new valuations (*i.e.,* neither in $\mathcal{H}$ nor in Q) based on the newly deduced fix $(t'[B], c)$ (by simply checking $\mathcal{S}[t'[B]]$) and add them to Q if their preconditions are validated. (2) If $\overline{U}$ is invalid (*i.e.,* $\{(t'[B], c_1), (t'[B], c_2)\} \subseteq \overline{U}$, but $c_1 \neq c_2$, lines 6-8), we call a procedure ResolveConflict (see below), to decide the true value c of $t'[B]$ and add $(t'[B], c)$ to Γ. Set $\overline{U}$ and affected valuations in Q and $\mathcal{H}$ are updated accordingly based on the true value of $t'[B]$, and the chase will be resumed. This process continues until Q is empty. Finally, we update Γ with the fixes (line 10) and return $\mathsf{TS}(t)$ (line 11).

Procedure ResolveConflict. Taking a tuple $t' \in \mathsf{TS}(t)$ and attribute B as input, ResolveConflict decides how to assign $t'[B]$, via correlation analysis. Assume the set of candidate values (*i.e.,* the active domain) for $t'[B]$ is $\{c_i \mid \exists\varphi \in \Sigma_a : X \rightarrow t'[B] = c_i\}$. We assign

$$t'[B] = \arg\max_{\forall c_i} \mathcal{M}_c(t'[\bar{A}], c_i),$$

where $t'[\bar{A}]$ is the validated partial tuple after the initial splitting. Intuitively, if c_i is strongly correlated with $t'[\bar{A}]$, we set $t'[B] = c_i$.

Example 5: Consider the chase in Example 4 with $\Sigma_a = \{\varphi_2, \varphi_3\}$. As ($t_b$[festCity], "Munich") is not validated when it starts, Q (resp. $\mathcal{S}[t_b[\text{festCity}]]$) is initialized as $\{h_2 : t_b \rightarrow t\}$ (resp. $\{h_3 : (t_b \rightarrow t)\}$). We first process h_2 in Q; it validates t_b[festCity] = "Munich". Then we check valuations in $\mathcal{S}[t_b[\text{festCity}]]$. When all predicates in the precondition of h_3 are validated, h_3 is added to Q. After deducing t_b[festCountry] by applying (φ_3, h_3), the chase terminates. □

Analysis. The correctness of Splitting is partially warranted by Corollary 1. Under certain assumptions on the ML models embedded in the REE$^+$s of Σ_a, it retains the Church-Rosser property with conflict resolution. Due to the space limit, the proofs are reported in [12]. Splitting takes $O(c_{\mathsf{valid}}|D||R||\Sigma_a|(\sum_{t\in D_V}|\mathsf{TS}(t)|)^2)$ time, where c_{valid} denotes the unit cost of validating a valuation for an REE$^+$. This is because the length of a chasing sequence is $O(|R|\sum_{t\in D_V}|\mathsf{TS}(t)|)$, and there are $|\Sigma_a|$ REE$^+$s in Σ_a; for each REE$^+$ φ in Σ_a, at most $O(|D|\sum_{t\in D_V}|\mathsf{TS}(t)|)$ valuations are checked.

6 COMPLETING SPLIT TUPLES

In this section we show how to complete tuples in $\mathsf{TS}(t)$ by imputing missing values (Section 6.1). In particular we train an ML model for suggesting values (Section 6.2). The method works for imputing incomplete information in general, not limited to tuple splitting.

6.1 Imputing Missing Values

We fill in the missing values of $\mathsf{TS}(t)$ by combining logic deduction, ML prediction and data extraction from knowledge graphs in a uniform logical framework by chasing $\mathsf{TS}(t)$ with a set Σ_c of REE$^+$s.

REE$^+$s. We use three types of REE$^+$s, prioritizing the first two:

(1) (Logic) Bi-variable REEs in [55, 57] of the form $X \rightarrow t[A] = c$, *e.g.,* REEs similar to φ_3 to deduce t_a[festCountry] = "Germany".
(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 the knowledge graph G and if x reaches vertex v via path ρ, which encodes the B-attribute of t', then $t'[B]$ takes the value (label) of v, *e.g.,* t_b[born] = "1982" by φ_5 of Example 2.
(3) (ML prediction) REE$^+$s $R(t') \wedge \mathsf{null}(t'[B]) \rightarrow t'[B] = \mathcal{M}_d(t'[\bar{A}], B)$, where $t'[\bar{A}]$ is a partial tuple with all validated values and $\mathcal{M}_d$ is a model, which suggests a value to fill in null $t'[B]$ (Section 6.2), *e.g.,* t_a[college] = "ZHdK" by φ_4 in Example 2.

The chase. We extend Splitting (Section 5.3) to complete tuples by chasing with the REE$^+$s of types (1) and (2), with the following modification: If the chasing is valid, for each tuple t in D_V, we check whether all null values of tuples in $\mathsf{TS}(t)$ requested by the user are imputed. If so, $\mathsf{TS}(t)$ is returned. Otherwise, we randomly select a null $t'[B]$ in $\mathsf{TS}(t)$, set $t'[B] = \mathcal{M}_d(t'[\bar{A}], B)$, where $\bar{A}$ includes all validated attributes in t', and chase $\mathsf{TS}(t)$ iteratively. We use $\mathcal{M}_d$ of Section 6.2 to suggest attribute values only after REE$^+$s for logic deduction and data extraction cannot determine a right value for $t'[B]$.

Example 6: Continuing the examples, we complete $\mathsf{TS}(t_s) = \{t_a, t_b\}$ with $\Sigma_c = \{\varphi_4, \varphi_5\}$. By applying Σ_c, we get t_a = ("Noemi Schneider", null, "1986", "ZHdK", "Sturm", null, null, null). Assume that we assign t_a[festCity] = "Landshut" via the prediction model $\mathcal{M}_d$. We then chase $\mathsf{TS}(t_s)$ again using REE$^+$s just like φ_3 to deduce t_a[festCountry] = "Germany". This process continues until all null values required by the user are imputed and $\mathsf{TS}(t_s)$ is returned. □

Complexity. The cost of completing tuples is also dominated by the chase. A similar analysis show that in total, it takes $O((|V| + |E|)^2 \sum_{t\in D_V} |\mathsf{TS}(t)| + c_{\mathsf{valid}}|D||R||\Sigma_c|(\sum_{t\in D_V} |\mathsf{TS}(t)|)^2)$ time, since (a) it takes $O((|V| + |E|)^2)$ time to perform HER [50] for each tuple in $\mathsf{TS}(t)$ of $t \in D_V$ on $G(V, E, L)$, and (b) imputing null values does not increase the worst-case complexity.

6.2 Prediction Model $\mathcal{M}_d$

We extend $\mathcal{M}_c$ to $\mathcal{M}_d$ for value imputation; it takes a partial tuple $t[\bar{A}]$ and an attribute B as input, and suggests a value for $t[B]$.

Model $\mathcal{M}_d$. The model suggests missing values by referencing a knowledge graph G, in two steps. It first retrieves a set Cand_B of candidate values for $t[B]$ from G. If Cand_B is nonempty, a ranking model is used to get a suggested value for $t[B]$. Otherwise, we propose a (optional) remedy strategy to predict a value for $t[B]$.

Candidate values retrieval. We retrieve the set Cand_B of candidate values for $t[B]$ via HER. Given a partial tuple $t[\bar{A}]$, an attribute B and a knowledge graph $G = (V, E, L)$, we first extract a set V_t of vertices in G that match $t[\bar{A}]$ via HER [50]. Then for each vertex v in V_t, we check each of its k-hop neighbors v' in G for a predefined bound k. Let ρ be a label path from v to v', $\mathsf{sim}(\cdot)$ be a similarity measure, *e.g.,* BERT-based function of [50], and τ be a predefined similarity threshold. If $\mathsf{sim}(\rho, B) \geq \tau$, we add $L(v')$ as a candidate to Cand_B.

Ranking Model. Next we train a ranking model to get the top-ranked value in Cand_B as the suggested value for $t[B]$. We reuse the lookup table Dict and $\mathsf{Encoder_G}$ in $\mathcal{M}_c$ for this purpose. Specifically, we transform $I_t^d = (t[\bar{A}], B)$ to a matrix $\mathbf{M}_{\mathrm{G}}^d$ as in $\mathcal{M}_c$. Then I_t^d is encoded as $\mathbf{I}_t^d = \mathsf{Encoder_G}(\mathbf{M}_{\mathrm{G}}^d)$. For each value c in Cand_B, we compute its graph embedding $\mathbf{c} = \mathsf{Dict}[c]$. To map the embeddings to the same latent space, we use two encoders $\mathsf{Encoder_I}$ and $\mathsf{Encoder_c}$:

$$\mathbf{E}[\bar{A}]{=}\mathsf{Encoder_I}(\mathbf{I}_t^d){=}\sigma(\mathsf{FC_I}(\mathbf{I}_t^d)),\quad \mathbf{E}[c]{=}\mathsf{Encoder_c}(\mathbf{c}){=}\sigma(\mathsf{FC_c}(\mathbf{c})),$$

where FC is the fully-connected layer, σ is the sigmoid function, and $\mathbf{E}[\bar{A}]$ (resp. $\mathbf{E}[c]$) denotes the final embedding for $t[\bar{A}]$ (resp. c).

Intuitively, if $\mathbf{E}[\bar{A}]$ is correlated to $\mathbf{E}[c]$, $t[B]$ is likely to take value c. By measuring correlation via dot product, $\langle\cdot,\cdot\rangle$, we have

$$\mathcal{M}_d(t[\bar{A}], B) = \arg\max_{c\in \mathsf{Cand}_B} \langle \mathbf{E}[\bar{A}], \mathbf{E}[c]\rangle.$$

To make up the lack of training data, we adopt pairwise ranking with triplet loss. Given a set $\mathcal{T}_d$ of N training examples of the form $(t[\bar{A}], c_1, c_2)$ (*i.e.,* c_1 is more related to $t[\bar{A}]$ than c_2), triplet loss is

$$\mathcal{L}_{\mathsf{pair}}(\mathcal{T}_d) = \frac{1}{N}\sum_{(t[\bar{A}],c_1,c_2)\in\mathcal{T}_d}(\max(\langle \mathbf{E}[\bar{A}], \mathbf{E}[c_1]\rangle - \langle \mathbf{E}[\bar{A}], \mathbf{E}[c_2]\rangle + \gamma, 0)),$$

where γ is a predefined hyperparameter and it denotes the margin between the two dot products $\langle \mathbf{E}[\bar{A}], \mathbf{E}[c_1]\rangle$ and $\langle \mathbf{E}[\bar{A}], \mathbf{E}[c_2]\rangle$.

(Optional) Remedy model. When Cand_B is empty, we train a remedy model to predict $t[B]$. We adopt sentenceBert [106] as our base model, which treats $t[\bar{A}]$ as a sequence and returns its embedding, denoted by $\mathbf{E}[\bar{A}] = \mathsf{sentenceBert}(t[\bar{A}])$. Similarly, given a value c in $\mathsf{dom}(B)$, where $\mathsf{dom}(B)$ is the active domain of B (all B-attribute values of tuples in D), we compute $\mathbf{E}[c] = \mathsf{sentenceBert}(c)$. Then we let $t[B] = \arg\max_{c\in\mathsf{dom}(B)}\langle \mathbf{E}[\bar{A}], \mathbf{E}[c]\rangle$. This strategy combines two sentenceBert with shared parameters. In inference, we adopt Faiss [76] to retrieve the top-1 value from $\mathsf{dom}(B)$. This step is optional, *e.g.,* a user may opt to retain the null values if Cand_B is empty, and SET only fills in null values requested by the user via $\mathcal{M}_d$.

Table 1. The tested real-life datasets

Datasets	$\lvert D_o \rvert$	$\lvert D \rvert$	# of real mismatches	# of tuples to correct
Citation [79]	51,485	22,826	207	1,028
College [4]	20,483	4,670	124	206
Person [10]	948,856	285,962	4,936	13,721
IMDB [16]	3,205,737	1,057,217	4,670	49,521

Table 2. Training time and statistic

Datasets	Training time (s)		#conflicts detected per REE$^+$		Total #conflicts detected	
	$\mathcal{M}_c$	$\mathcal{M}_d$	for splitting	for correcting	for splitting	for correcting
Citation [79]	465s	297s	31	163	1,178	6,194
College [4]	93s	113s	11	23	451	943
Person [10]	703s	3245s	336	811	15,120	36,495
IMDB [16]	1402s	4972s	353	6,663	10,943	206,553

7 EXPERIMENTAL STUDY

Using real-life datasets, we experimentally evaluated (1) the effectiveness of SET for deciding what tuples to split and what tuples to correct (DS), assigning attribute value and correcting errors (AA), and missing value imputation (MI); (2) the efficiency of SET (DS, AA and MI); and (3) the use of SET in real life via a case study.

Experimental settings. We start with our experimental settings.

Datasets. We used four real-life datasets D_o: (1) Citation [79], an extended ER benchmark of citations from DBLP and ACM. We enriched the schema with 7 more attributes in DBLP RDF data [6] by using a predefined mapping function, and expanded Citation with more tuples if the mapping is one-to-many. (2) College [4], a dataset of colleges in the USA. Following [88, 99, 120], we enlarged College so that there is enough training data. (3) Person [10], a dataset of person tuples crawled from Wikipedia, and (4) IMDB [16], a set of movies and TV Series released between 1905 and 2022.

The set D of target tuples has two parts. (1) The set of merged tuples obtained by merging tuples in D_o via a state-of-the-art ER model ditto [88]. We fed pairs of tuples to ditto. If ditto predicts true, we merged the tuples into one and added it to D. If there is a conflict for an attribute (*e.g.,* VLDB and VLDBJ), we randomly picked one value so that mismatches are non-trivial to identify. The set of real mismatches is the subset of merged tuples that are predicted true by ditto but they represent different entities (need to split). (2) A subset of randomly selected tuples from D_o whose size is about 5% of the tuples in (1), with injected errors (*i.e.,* our error ratio $\approx$ 5% [92, 108]). Here errors are injected by modifying two attributes of each tuple with values in their domains as errors (need to correct).

Table 1 shows the number $\lvert D_o \rvert$ of tuples, the number $\lvert D \rvert$ of target tuples (including both tuples from D_o and merged tuples that are predicted positive by ditto), the number of real mismatches (false positives of ditto to be split) and the number of tuples with errors to be corrected without splitting. To better visualize the effect, we mainly focus on the merge of a pair of tuples in most experiments.

We used widely adopted KGs in benchmarks as G: (1) DBLP RDF [6] for Citation; (2) college data from National Center for Education Statistics [5] (transformed to RDF) for College; (3) Wikidata [21] for Person; and (4) the officially released movie dataset [16] for IMDB.

Models and data extraction. We trained $\mathcal{M}_c$ and $\mathcal{M}_d$ (resp. ditto) with 20% (resp. 10%) of tuples and used the remaining for testing. We used graph embeddings of 200-dimension with PyTorch-BigGraph [84], and pretrained Bert [44] (bert_en_uncased). The learning rate for $\mathcal{M}_c$ and $\mathcal{M}_d$ (resp. ditto) is 5e-4 (resp. 3e-5). We adopted batch sizes 256, 128 and 256 for $\mathcal{M}_c$, $\mathcal{M}_d$ and ditto with epochs 150, 100, 10, respectively. We adopted JedAI [103] as HER for its popularity. We report the training time of $\mathcal{M}_c$ and $\mathcal{M}_d$ on all datasets in Table 2: the training time of $\mathcal{M}_c$ is comparable to ditto in [88], while $\mathcal{M}_d$ takes longer since it has to handle a more complex ranking task.

Baselines. We implemented SET in Python [12]. We used the following baselines. (1) Bert [44], an ML approach that treats DS and MI as downstream tasks of pretrained Bert, such that DS (resp. MI) is conducted as a ternary classifier (resp. a remedy model), while AA is based on the confidences of DS. (2) Raran, a hybrid ML error detection and correction method that adopts Raha [92] for error detection, and uses Baran [91] for correction. Here Raha also complies KB rules from G for detecting violations. (3) Holoclean [108], a hybrid data repairing method that integrates DCs [26] (also used for error detection), external information (*i.e.,* the ground truth Γ) and statistics (*i.e.,* frequency). (4) Imp3C [45], which conducted data repairing based on CFD deduction on the ground truth Γ, and naive Bayes.

We also compared the following variants of SET: (5) SET_{noML}, which adopts only REE^+s without ML predicates $\mathcal{M}_c$ and $\mathcal{M}_d$. (6) $\text{SET}_{\text{noHER}}$, which does not support HER; note that without HER, $\mathcal{M}_d$ of $\text{SET}_{\text{noHER}}$ (which extracts candidate values via HER) is reduced to a remedy model (see Section 6). (7) SET_{NC}, which adopts the brute-force methods for the chase, via match enumeration. Since SET and SET_{NC} produce the same results, we compared with SET_{NC} only for efficiency, and with other baselines for effectiveness.

Note that no prior systems support tuple splitting. Holoclean, Imp3C and Raran only detect and correct errors. Nevertheless, these methods were evaluated in the tuple splitting setting for which they were not designed since they only aim to correct individual tuples. We extended Bert for both tuple splitting and error correcting.

*Rules, ground truth and labels.*We mined 38, 41, 45 and 31 REE^+s on Citation, College, Person and IMDB, respectively (Section 3). We report the number of conflicts detected (per REE^+), for splitting tuples and correcting tuples in Table 2. Note that multiple conflicts can be detected on the attributes of a single tuple by different REE^+.

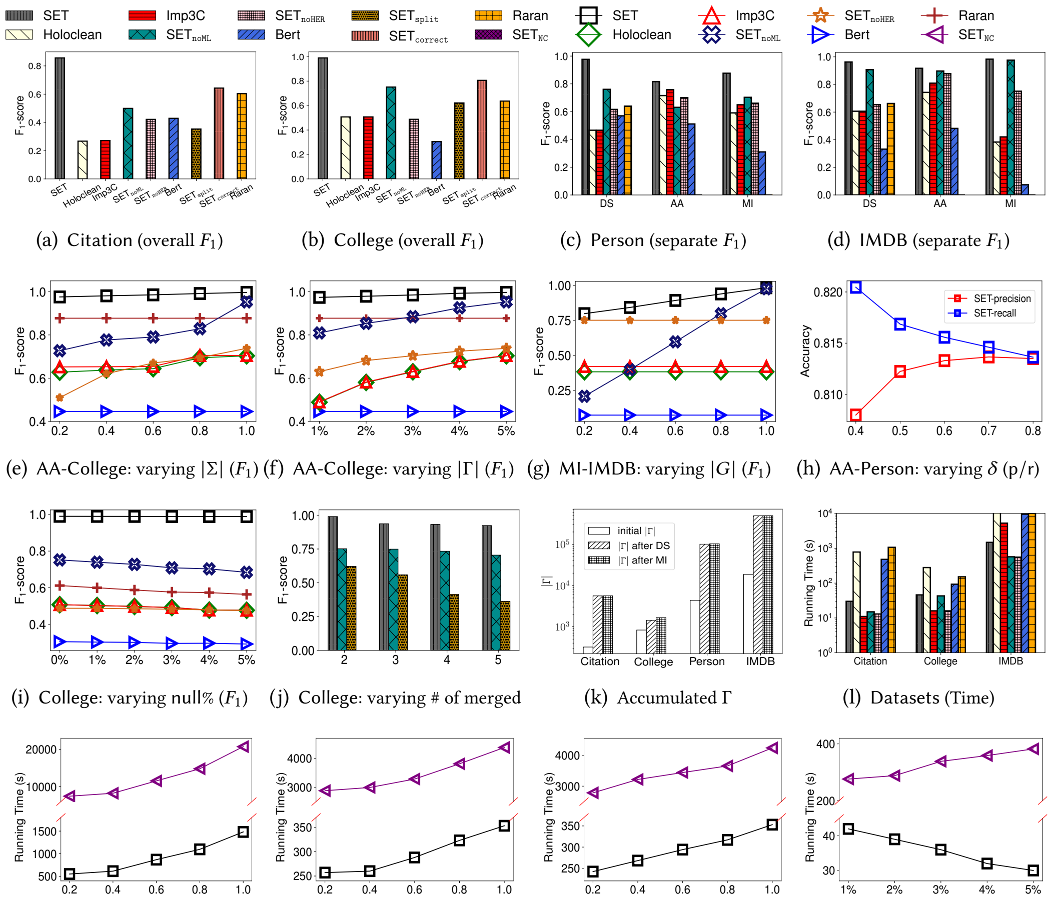
For initial ground truth Γ, we randomly sampled 5% tuples from each dataset and validated the facts in its corresponding knowledge graph G; we provided HoloClean and Imp3C with the same Γ, which is gradually accumulated during the chase. To be fair, we also set the labeling budget of Raran to be 5% of data in the dataset.

Configuration. We conducted the experiments on a single machine powered by 256GB RAM and 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz. Each test was run 3 times; the average is reported here. For the lack of space we report results on some datasets; the results on the others are consistent (reported in [12]).

Experimental results. We next report our findings.

Exp-1 Effectiveness. We first evaluated the overall accuracy (including DS, AA and MI) using $F_1\text{-score} = 2 \times \frac{\text{recall}\times\text{precision}}{\text{recall}+\text{precision}}$, where recall is the ratio of tuples we correctly rectify, split and impute to all tuples that need to be corrected, split and imputed, and precision is the ratio of correctly rectified, split and imputed tuples to all tuples we correct, split and impute. Unless stated explicitly, the default scaling factors (resp. sampling ratio) of G, the set Σ of REE^+s and the set D of target tuples (resp. the initial Γ) are all 100% (resp. 5%). We set $\delta = 0.5$ and $k = 1$, where δ and k denote the threshold of $\mathcal{M}_c$ and the number of hops for $\mathcal{M}_d$, respectively. To efficiently evaluate whether split tuples are unique entities, we built a tree-based index structure on G to support fast search and checking. The accuracy of error correction in Raran is not reported on IMDB and Person since their generated features are too large to fit in memory.

Accuracy vs. baselines. Denote by $\text{SET}_{\text{split}}$ (resp. $\text{SET}_{\text{correct}}$) when SET is only used to split tuples



(a) Citation (overall F_1) (b) College (overall F_1) (c) Person (separate F_1) (d) IMDB (separate F_1)

(e) AA-College: varying $|\Sigma|$ (F_1) (f) AA-College: varying $|\Gamma|$ (F_1) (g) MI-IMDB: varying $|G|$ (F_1) (h) AA-Person: varying δ (p/r)

(i) College: varying null% (F_1) (j) College: varying # of merged (k) Accumulated Γ (l) Datasets (Time)

(m) IMDB: varying $|D|$ (Time) (n) Person: varying $|G|$ (Time) (o) Person: varying $|\Sigma|$ (Time) (p) Citation: varying $|\Gamma|$ (Time)

Fig. 5. Performance evaluation

(resp. correct errors). As shown in Figures 5(a) and 5(b) on Citation and College, respectively SET has the best F_1-score on both datasets. Besides, we find the following.

(1) As expected, SET is 43.5% and 19.8% more accurate than SET$_{split}$ and SET$_{correct}$ on average, respectively. This shows SET$_{correct}$ or SET$_{split}$ alone does not suffice for improving the overall data quality, since they solve a ternary problem (*i.e.*, distinguish tuples to split, tuples to correct, and tuples that need neither split nor correction) using binary classification (*e.g.*, SET$_{correct}$ decides whether it corrects a tuple or not). Consider a mismatch of two entities. SET$_{correct}$ treats the true values from one entity as errors, and corrects them with the information from the other entity. No matter which entity SET$_{correct}$ chooses to fix, it loses the information of the other entity, which inevitably affects the accuracy. Similar explanation applies to existing data repairing methods that focus on error correction (see below).

(2) SET beats the baselines, *e.g.*, on average its F_1-score is 0.92, as opposed to 0.389, the best of rule-based methods Holoclean and Imp3C, and 0.607, the best of ML-based methods Raran and Bert. If we focus on correcting errors alone (the binary problem), Raran performs as the best repairing-based baseline and is comparable to SET$_{correct}$, since both of them can benefit from G and the ground truth.

However, when it comes to tuple splitting (a ternary problem), its performance inevitably degrades, due to similar reasons above. Worse still, the inaccuracy is propagated and magnified along the splitting process, from DS to AA to MI, leading to even worse overall accuracy. This explains why existing data repairing techniques do not suffice for solving the tuple splitting problem.

(3) We evaluated the separate accuracy for DS, AA and MI in Figures 5(c)-5(d). We focus on F_1-scores on each task, *e.g.,* for DS, the F_1-score measures tuples that SET correctly decides to fix/split. SET consistently beats the baselines, *e.g.,* its F_1-score is 31.8%, 8.3% and 39.5% higher than the best of the baselines in the three, respectively. This is because SET simultaneously corrects errors and splits tuples by combining rules, ML correlation and data extraction, while the baselines only correct conflicts, and only use either rules or ML models; this also verifies the need for splitting. SET not only predicts correlation but also deduces reliable values with Γ and G, and accumulates ground truth. Moreover, its F_1-score is 57.9% better than Bert on average, showing the effectiveness of unifying logic and ML.

Compared with its variants, SET also does the best in each stage, *e.g.,* its F_1-score is 13.5% higher than SET_{noML} on DS, and 22.4% higher than $\text{SET}_{\text{noHER}}$ on MI. Note that SET_{noML} is sometimes as good as SET on MI, when HER is accurate enough to impute missing values via data extraction from knowledge graphs.

We next tested the impact of various parameters on the accuracy.

Varying $|\Sigma|$. We varied $|\Sigma|$ from 20% to 100%. We focus on AA, and the trend of DS and MI is consistent. As shown in Figure 5(e), (1) SET gets more accurate when given more REE^+s, *e.g.,* its F_1-score increases by 2.1% on College for AA. This is because more REE^+s have more valuations, and more values can be correctly assigned/fixed by SET. (2) SET beats HoloClean and Imp3C by 34.8% and 32.3% on average, respectively, even with only 20% rules. This verifies the effectiveness of REE^+s that support correlation and extraction predicates for error correction and missing value imputation (not shown). (3) With 20% of rules, SET beats ML-based Raran and Bert by 10.8% and 52.9%, respectively, since SET utilizes correlation analysis, which is particularly useful for the tuple splitting problem.

SET beats its variants. On average, (a) it is 17% more accurate than SET_{noML}. This again verifies the need for correlation analysis in splitting or correcting tuples. (b) Its F_1-score is 33.9% higher than $\text{SET}_{\text{noHER}}$, since SET references knowledge graphs to decide what tuples to split/correct, and generates more certain fixes. (c) The increasing treads of SET are less obvious than its variants, since logic rules, ML models and data extraction complement each other.

Varying $|\Gamma|$. As shown in Figure 5(f) by varying the sampling ratio of initial ground truth Γ from 1% to 5%, (1) SET has a higher F_1-score when $|\Gamma|$ gets larger, as expected, *e.g.,* the F_1-score of SET is improved by 2.3% on College for AA. (2) SET performs the best; its F_1-score beats the best of the baselines, Raran, by 10.8%. (3) With only 1% of ground truth, SET outperforms the baselines by 39.8% on average. This verifies that SET takes good advantage of ground truth. Similarly the sampling ratio of Γ affects DS and MI.

Varying $|G|$. We varied the size $|G|$ of knowledge graphs that can be referenced from 20% to 100% in Figure 5(g); this is translated to 20% to 100% of graph embeddings that can be referenced by $\mathcal{M}_c$ and $\mathcal{M}_d$. As expected, the F_1-score of SET is improved by 18.5% for MI when $|G|$ is from 20% to 100%. This is because (a) HER could extract more information from G, and (b) $\mathcal{M}_d$ could get richer embeddings.

Varying δ. Varying the threshold δ of $\mathcal{M}_c$ from 0.4 to 0.8, we evaluated its impact on accuracy in Figure 5(h). To illustrate better, we report precision and recall instead of F_1-score. Note that δ affects DS and AA differently. (a) For AA, a value is assigned to t if it is strongly correlated with the existing values in t (*i.e.,* $\mathcal{M}_c(t[\bar{A}], t[B]) \geq \delta$). With larger δ, less values could be assigned and hence recall gets smaller. (b) In contrast, the assignment is more likely to be correct when δ increases, and precision gets higher. DS is affected conversely. To strike a balance, we set $\delta = 0.5$ by default.

Varying null%. To test the impact of null values, we also evaluated SET by varying the ratio of null values (null%) in D to be filled in, from 1% to 5% in Figure 5(i). As expected, the accuracy of SET degrades slightly when more null values need to be filled in. This said, SET still performs the best compared with all the baselines.

Varying #tuples *merged.* We varied the # of tuples that we merged into one in Figure 5(j), where we only compared SET and its variants. When more tuples are merged, it is more challenging to split them, and the accuracy gets a bit lower. Nevertheless, the F_1-score of SET is still as high as 0.936 when 3 tuples in College are merged into one.

Accumulated Γ. We report in Figure 5(k) the size of Γ of ground truth accumulated during the process of tuple splitting and error correcting, given 1% initial ground truth. As shown there, $|\Gamma|$ gets larger after each task; it starts with 832 validated tuples in Γ on College; then $|\Gamma|$ is increased 1.7-fold (resp. 2-fold) after DS (resp. MI).

We also tested the impact of hop number k of $\mathcal{M}_d$ on MI (not shown). We find that the accuracy of SET is not sensitive to k since most values can be found in 1 hop; it takes longer with larger k since more vertices in G have to be checked. Thus we set $k = 1$ by default.

Exp-2: Efficiency. We first compared the efficiency of SET and the baselines in the default settings. To be fair, the ML training time of the baselines is excluded. Then we compared the efficiency of SET and SET_{NC} to justify our implementation of chase. Note that we do not report the time of a baseline if it could not finish within 3 hours. Similarly, we checked G to decide whether and how to split tuples with conflicts using the tree-based index on G.

Efficiency vs. baselines. As shown in Figure 5(l) on three real-life datasets, the time of SET is comparable with or slightly slower than rule-based methods (*e.g.,* Imp3C) in most cases, but is much faster than ML-based methods, *e.g.,* SET takes 46s to execute on College, which is 2X and 3.3X faster than Bert and Raran, respectively.

Efficiency vs. SET_{NC}. Figures 5(m) to 5(p) compare SET and SET_{NC}.

(1) Varying $|D|$. Varying $|D|$ from 20% to 100%, we report the total time in Figure 5(m). As expected, both methods run slower when $|D|$ increases, since more tuples need to be checked. Nonetheless, SET is 13.5X faster than SET_{NC} on average. This shows that maintaining partial chasing results avoids costly enumeration. SET is efficient: it takes 1,481s when D has 1,057,217 tuples.

(2) Varying $|G|$. We varied knowledge graph $|G|$ from 20% to 100% in Figure 5(n). Both methods take longer with larger G since they check more. SET is still 11.6X faster than SET_{NC} on average.

(3) Varying $|\Sigma|$. We varied $|\Sigma|$ from 20% to 100%. As reported in Figure 5(o), both SET and SET_{NC} take longer when given more REE^+s, as expected, since it needs more time to process the valuations when given more REE^+s. SET is 11.8X faster than SET_{NC} on average.

(4) Varying $|\Gamma|$. Varying sampling ratio of ground truth from 1% to 5% in Figure 5(p), SET takes less time, from 42s to 30s. This is because the time of SET is dominated by ML prediction and data extraction, while logic deduction is fast. With larger Γ, more mismatches (resp. conflict/missing values) can be identified (resp. corrected/imputed) by REE^+s, leaving less work to ML and HER and thus, the overall runtime is reduced. However, SET_{NC} does not behave similarly, since its cost is dominated by costly valuation enumeration. SET consistently beats SET_{NC}, *e.g.,* it is 9.1X faster on average.

Exp-3. Case study. We crawled a person dataset with schema R = (name, DoB, death, occupation, party) from Wikipedia [7]. We obtained person tuples based on names (*e.g.,* Hirai Tarō [7]) and cited Wiki pages (*e.g.,* [9]). From the tuples, SET found a *real mismatch*, represented by t = ("Hirai

Tarō", "21/10/1894", "28/07/1965", "Novelist", "Liberal democratic party"). Here t is a mismatch from a famous Japanese novelist and a Japanese councilor (see the erroneous link to "Hirai Tarō" in [9]). To correct it, SET splits t as follows.

(1) DS. By applying REE$^+$ φ_8 : $R(t) \wedge \mathcal{M}_c(t[\text{occupation}], t[\text{party}]) \leq 0.2 \rightarrow$ false, SET finds the correlation between $t[\text{occupation}]$ and $t[\text{party}]$ is low (*e.g.,* 0.1). Thus, t is an abnormal tuple that needs to be split/corrected; party is its conflicting attribute. We reference Wikidata [21] G by HER and confirm that t is a mismatched tuple of distinct entities (*i.e.,* $f(t) \notin \mathcal{E}$). We then (initially) split $\text{TS}(t)$ into $\{t_a, t_b\}$, where t_a = ("Hirai Tarō", null, null, "Novelist", null) and t_b = ("Hirai Tarō", null, null, null, "Liberal democratic party").

(2) AA. To distribute "21/10/1894" of $t[\text{DoB}]$, we apply φ_9 : $R(t) \wedge \mathcal{M}_c(t[\text{name, occupation}], t[\text{DoB}] = \text{"21/10/1894"}) \geq 0.6 \rightarrow t[\text{DoB}] = \text{"21/10/1894"}$ via valuation $h_9 : t_a \rightarrow t$ of φ_9. Since $\mathcal{M}_c(t_a[\text{name, occupation}], \text{"21/10/1894"}) = 0.8 > 0.6$, "21/10/1894" is strongly correlated with t_a; thus we assign it to $t_a[\text{DoB}]$ by φ_9. Similarly, we assign "28/07/1965" of $t[\text{death}]$ to $t_a[\text{death}]$.

(3) MI. To impute null values in $\text{TS}(t)$, *e.g.,* $t_b[\text{occupation}]$, SET uses Wikidata as external knowledge graph G and applies φ_{10} : $R(t) \wedge \text{vertex}(x, \text{Wikidata}) \wedge \text{HER}(t, x) \wedge \text{match}(t[\text{occupation}], x.(\text{occupation})) \rightarrow t[\text{occupation}] = \text{val}(x.\text{occupation})$, setting $t_b[\text{occupation}]$ = "Councilor". After filling in all requested null values, the splitting of t is done, with t_a = ("Hirai Tarō", "21/10/1894", "28/07/1965", "Novelist", "Nonparty") and t_b = ("Hirai Tarō", "17/07/1905", "04/12/1973", "Councilor", "Liberal democratic party").

Summary. We find the following. (1) By unifying logic deduction, ML correlation and data extraction, SET is the most accurate for the overall tuple splitting problem, *e.g.,* 0.92 F_1-score on average as opposed to 0.607, 0.387 and 0.389 by repairing-based Raran, HoloClean and Imp3C, and 0.428 by ML-based Bert. (2) SET consistently outperforms the baselines in DS, AA and MI, *e.g.,* its F_1-score is 31.8%, 8.3% and 39.5% higher than the best of baselines on average, respectively. (3) SET outperforms its variants SET_{noML} and $\text{SET}_{\text{noHER}}$ in accuracy by 22.6% on average. This justifies the need for ML correlation model and data extraction. (4) SET is 19.8% more accurate than $\text{SET}_{\text{correct}}$ on average; this justifies the need for tuple splitting. (5) Tuple splitting/completing with the chase is efficient by maintaining partial results; it reduces the total time from 20,734s to 1,481s on IMDB.

8 CONCLUSION

The novelty of the work consists of (1) a new problem for tuple splitting; (2) an extension of error correction with tuple splitting, by unifying logic, ML and data extraction in the same process; (3) ML models to assess correlation among attributes and predict missing values; (4) extended REEs to support correlation models, heterogeneous ER and data extraction; and (5) algorithms for identifying tuples of mismatched entities, splitting tuples, deducing certain fixes and imputing missing values with various REE$^+$s. Our experimental study has shown that SET is promising in practice.

One topic for future work is to extend SET for imputing both missing values and missing tuples. Another topic is incremental splitting in response to updates. A third topic is to study the impact of bias in datasets on correlation model $\mathcal{M}_c$ and overall accuracy.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2021ZD0113903), Royal Society Wolfson Research Merit Award WRM/R1/180014, NSFC 62202313, NSFC 62225202, Guangdong Basic and Applied Basic Research Foundation 2022A1515010120 and Longhua Science and Technology Innovation Bureau 10162A20220720B12AB12.

REFERENCES

[1] 2013. Lego Friends. https://www.imdb.com/title/tt4049416/.
[2] 2013. Lego Friends. https://www.imdb.com/title/tt9148446/.
[3] 2013. Storm. http://filmstudieren.ch/en/storm#1.
[4] 2022. Colleges. https://data.world/dhs/colleges-and-universities.
[5] 2022. Colleges KG. https://nces.ed.gov/GLOBALLOCATOR/.
[6] 2022. DBLP. https://dblp.org/rdf/release/dblp-2022-05-02.nt.gz.
[7] 2022. Elected Councillors in Kagawa at-large district of Japan. https://en.wikipedia.org/?curid=27298128.
[8] 2022. Help:Conflation of two people. https://www.wikidata.org/wiki/Help:Conflation_of_two_people.
[9] 2022. Hirai Tarō (novelist). https://en.wikipedia.org/wiki/Edogawa_Ranpo.
[10] 2022. Wikemedia. https://www.kaggle.com/datasets/kenshoresearch/kensho-derived-wikimedia-data.
[11] 2023. BA film. https://www.zhdk.ch/en/degree-programmes/film/ba-film.
[12] 2023. Code, datasets and full version. https://drive.google.com/drive/folders/1-Bc20q3hc26cqW-7zJ3R0xHm-t00CrIu?usp=sharing.
[13] 2023. DOK.fest. https://www.dokfest-muenchen.de/.
[14] 2023. Dun & Bradstreet. https://www.dnb.com/.
[15] 2023. Filmography by ZHdK. https://www.swissfilms.ch/en/company/zrcher-hochschule-der-knste-zhdk-departement-darstellende-knste-und-film/A96DAF3F0CF04DEDBD79404DC793ED02.
[16] 2023. IMDB. https://www.imdb.com/interfaces/.
[17] 2023. IMDB Name Split. https://help.imdb.com/article/contribution/names-biographical-data/names/GSA3M6SFHRAERXZ3#.
[18] 2023. Noemi Schneide (German). https://www.dokfest-muenchen.de/films/walaa?lang=en & https://de.wikipedia.org/wiki/Noemi_Schneider.
[19] 2023. Noemi Schneider (Swiss). https://www.swissfilms.ch/en/person/nomi-natascha-schneider/385CEC7054A64FDC946F008A4432A4B9.
[20] 2023. US Bureau of Labor Statistics. https://www.bls.gov/.
[21] 2023. Wikidata. https://www.wikidata.org.
[22] 2023. Wikipedia. https://en.wikipedia.org/.
[23] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[24] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.
[25] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*. 952–963.
[26] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
[27] Zeinab Bahmani, Leopoldo E. Bertossi, and Nikolaos Vasiloglou. 2017. ERBlox: Combining matching dependencies with machine learning for entity resolution. *Int. J. Approx. Reasoning* 83 (2017), 118–141.
[28] Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. 2021. Missing Value Imputation on Multidimensional Time Series. *PVLDB* 14, 11 (2021), 2533–2545.
[29] Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering.* Morgan & Claypool Publishers.
[30] 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.
[31] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *J. Mach. Learn. Res.* 20, 175 (2019), 1–6.
[32] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. 2006. Adaptive Blocking: Learning to Scale Up Record Linkage. In *ICDM*. 87–96.
[33] Cory Bohon. 2022. How to find and merge duplicate contacts in iOS 16. https://www.techrepublic.com/article/merge-duplicate-contacts-ios-16/.
[34] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*. 1247–1250.
[35] Zhaoqiang Chen, Qun Chen, Boyi Hou, Murtadha Ahmed, and Zhanhuai Li. 2018. Improving machine-based entity resolution with limited human effort: A risk perspective. In *International Workshop on Real-Time Business Intelligence and Analytics.* 1–5.

[36] Zhaoqiang Chen, Qun Chen, Boyi Hou, Zhanhuai Li, and Guoliang Li. 2020. Towards interpretable and learnable risk analysis for entity resolution. In *SIGMOD*. 1165–1180.

[37] E. F. Codd. 1972. Relational Completeness of Data Base Sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California* (1972).

[38] Jess Cody. 2022. Where does data come from. https://clearbit.com/blog/where-does-data-come-from.

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

[40] 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.

[41] Sushovan De, Yuheng Hu, Venkata Vamsikrishna Meduri, Yi Chen, and Subbarao Kambhampati. 2015. BayesWipe: A Scalable Probabilistic Framework for Cleaning BigData. *CoRR* abs/1506.08908 (2015).

[42] Ting Deng, Wenfei Fan, Ping Lu, Xiaomeng Luo, Xiaoke Zhu, and Wanhe An. 2022. Deep and Collective Entity Resolution in Parallel. In *ICDE*. IEEE, 2060–2072.

[43] Daniel Deutch, Nave Frost, Amir Gilad, and Oren Sheffer. 2021. Explanations for Data Repair Through Shapley Values. In *CIKM*. ACM.

[44] 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.

[45] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2020. Leveraging currency for repairing inconsistent and incomplete data. *TKDE* (2020).

[46] 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.

[47] e_kartoffel. 2015. Names merged in error (by me). https://community-imdb.sprinklr.com/conversations/data-issues-policy-discussions/names-merged-in-error-by-me/5f4a79838815453dba7fbebc.

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

[49] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[50] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugey, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE, 634–647.

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

[52] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2023. Discovering Top-k Rules using Subjective and Objective Criteria. In *SIGMOD*. ACM.

[53] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2011. CerFix: A System for Cleaning Data with Certain Fixes. *PVLDB* 4, 12 (2011), 1375–1378.

[54] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *VLDBJ* 21, 2 (2012), 213–238.

[55] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

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

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

[58] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *IJCAI*. 4961–4967.

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

[60] Erdun Gao, Ignavier Ng, Mingming Gong, Li Shen, Wei Huang, Tongliang Liu, Kun Zhang, and Howard D. Bondell. 2022. MissDAG: Causal Discovery in the Presence of Missing Data with Continuous Additive Noise Models. In *NeurIPS*.

[61] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *PVLDB* 6, 9 (2013), 625–636.

[62] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning denial constraint violations through relaxation. In *SIGMOD*. 805–815.

[63] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On multiple semantics for declarative database repairs. In *SIGMOD*.

817–831.
[64] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude W. Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*. ACM.
[65] Lovedeep Gondara and Ke Wang. 2017. Multiple imputation using deep denoising autoencoders. *arXiv preprint arXiv:1705.02737* 280 (2017).
[66] Songtao Guo, Xin Luna Dong, Divesh Srivastava, and Remi Zajac. 2010. Record Linkage with Uniqueness Constraints and Erroneous Values. *PVLDB* 3, 1 (2010), 417–428.
[67] Shuang Hao, Nan Tang, Guoliang Li, Jianhua Feng, and Ning Wang. 2021. Mis-categorized entities detection. *VLDB J.* 30, 4 (2021), 515–536.
[68] IMDb help center. 2023. How can I combine two IMDb name pages? https://help.imdb.com/article/contribution/names-biographical-data/how-can-i-combine-two-imdb-name-pages/G3TNPWSGKZNRU3MP?ref_=helpsrall#.
[69] Benjamin Hilprecht and Carsten Binnig. 2021. ReStore - Neural Data Completion for Relational Databases. In *SIGMOD*. 710–722.
[70] Vinh Thinh Ho, Daria Stepanova, Mohamed H Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. 2018. Rule learning from knowledge graphs guided by embedding models. In *ISWC*. Springer, 72–90.
[71] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[72] Boyi Hou, Qun Chen, Zhaoqiang Chen, Youcef Nafa, and Zhanhuai Li. 2018. r-HUMO: A Risk-aware Human-Machine Cooperation Framework for Entity Resolution with Quality Guarantees. *TKDE* 32, 2 (2018), 347–359.
[73] Huldra. 2020. Help talk: Conflation of two people. https://www.wikidata.org/wiki/Help_talk:Conflation_of_two_people.
[74] Vassilis N Ioannidis, Xiang Song, Saurav Manchanda, Mufei Li, Xiaoqin Pan, Da Zheng, Xia Ning, Xiangxiang Zeng, and George Karypis. 2020. Drkg-drug repurposing knowledge graph for covid-19. https://github.com/gnn4dr/DRKG/.
[75] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk server-adding missing links while consuming linked data. In *COLD*. 85–96.
[76] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[77] Mourad Khayati, Ines Arous, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams. *PVLDB* 14, 3 (2020), 294–306.
[78] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. Mind the gap: An experimental evaluation of imputation of missing values techniques in time series. In *PVLDB*, Vol. 13. 768–782.
[79] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1-2 (2010), 484–493.
[80] loannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.
[81] Trent Kyono, Yao Zhang, Alexis Bellot, and Mihaela van der Schaar. 2021. MIRACLE: Causally-Aware Imputation via Learning Missing Data Mechanisms. In *NeurIPS*. 23806–23817.
[82] Dongjin Lee and Kijung Shin. 2021. Robust factorization of real-world tensor streams with patterns, missing values, and outliers. In *ICDE*. IEEE, 840–851.
[83] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* (2015).
[84] Adam Lerer, Ledell Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-BigGraph: A Large Scale Graph Embedding System. In *MLSys*.
[85] Alexander K. Lew, Monica Agrawal, David A. Sontag, and Vikash K. Mansinghka. 2020. PClean: Bayesian Data Cleaning at Scale with Domain-Specific Probabilistic Programming. *CoRR* abs/2007.11838 (2020).
[86] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks.. In *AAAI*. 8172–8179.
[87] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and Analysis]. *CoRR* abs/1904.09483 (2019).
[88] 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.
[89] Xi Liang, Zechao Shang, Sanjay Krishnan, Aaron J Elmore, and Michael J Franklin. 2020. Fast and reliable missing data contingency analysis with predicate-constraints. In *SIGMOD*. 285–295.
[90] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2020. Picket: Self-supervised Data Diagnostics for ML Pipelines. *CoRR* abs/2006.04730 (2020).
[91] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *PVLDB* 13, 12 (2020), 1948–1961.

[92] 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.

[93] Pierre-Alexandre Mattei and Jes Frellsen. 2019. MIWAE: Deep generative modelling and imputation of incomplete data sets. In *ICML*. PMLR, 4413–4423.

[94] John T McCoy, Steve Kroon, and Lidia Auret. 2018. Variational autoencoders for missing data imputation with application to a simulated milling circuit. *IFAC-PapersOnLine* 51, 21 (2018), 141–146.

[95] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A comprehensive benchmark framework for active learning methods in entity matching. In *SIGMOD*. 1133–1147.

[96] Yinan Mei, Shaoxu Song, Chenguang Fang, Haifeng Yang, Jingyun Fang, and Jiang Long. 2021. Capturing Semantics for Imputation with Pre-trained Language Models. In *ICDE*. IEEE, 61–72.

[97] Xiaoye Miao, Yangyang Wu, Lu Chen, Yunjun Gao, Jun Wang, and Jianwei Yin. 2021. Efficient and effective data imputation with influence functions. *PVLDB* 15, 3 (2021), 624–632.

[98] Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. 2021. Generative semi-supervised learning for multivariate time series imputation. In *AAAI*, Vol. 35. 8983–8991.

[99] 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.

[100] Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. 2020. Missing data imputation using optimal transport. In *International Conference on Machine Learning*. PMLR, 7130–7140.

[101] Alfredo Nazabal, Pablo M Olmos, Zoubin Ghahramani, and Isabel Valera. 2020. Handling incomplete heterogeneous data using vaes. *Pattern Recognition* 107 (2020), 107501.

[102] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. 2019. An embedding-based approach to rule learning in knowledge graphs. *TKDE* 33, 4 (2019), 1348–1359.

[103] 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.

[104] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.

[105] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *CIKM*. 1379–1388.

[106] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. 3980–3990.

[107] Florian Reitz. 2020. Corrections in dblp. https://blog.dblp.org/2020/01/08/corrections-in-dblp-2019/.

[108] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.

[109] Weilong Ren, Xiang Lian, and Kambiz Ghazinour. 2021. Online Topic-Aware Entity Resolution Over Incomplete Data Streams. In *SIGMOD*. 1478–1490.

[110] 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.

[111] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *SIGMOD*.

[112] Philipp Schirmer, Thorsten Papenbrock, Ioannis K. Koumarelas, and Felix Naumann. 2020. Efficient Discovery of Matching Dependencies. *ACM Trans. Database Syst.* (2020).

[113] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *PVLDB* 11, 2 (2017), 189–202.

[114] Qi Song, Peng Lin, Hanchao Ma, and Yinghui Wu. 2021. Explaining Missing Data in Graphs: A Constraint-based Approach. In *ICDE*. IEEE, 1476–1487.

[115] Shaoxu Song and Lei Chen. 2013. Efficient discovery of similarity constraints for matching dependencies. *Data Knowl. Eng.* 87 (2013), 146–166.

[116] Shaoxu Song, Yu Sun, Aoqian Zhang, Lei Chen, and Jianmin Wang. 2018. Enriching data imputation under similarity rule constraints. *TKDE* 32, 2 (2018), 275–287.

[117] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* 129 (2020), 249–260.

[118] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *WWW*.

[119] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *Int. J. of uncertainty, fuzziness and knowledge-*

based systems 10, 05 (2002), 557–570.
[120] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: A design space exploration. *PVLDB* 14, 11 (2021), 2459–2472.
[121] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.
[122] Steven Euijong Whang, Omar Benjelloun, and Hector Garcia-Molina. 2009. Generic entity resolution with negative rules. *VLDB J.* 18, 6 (2009), 1261–1277.
[123] Steven Euijong Whang and Hector Garcia-Molina. 2013. Joint entity resolution on multiple datasets. *VLDB J.* 22, 6 (2013), 773–795.
[124] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD*. 1149–1164.
[125] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *MLSys 2020*.
[126] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't Be SCAREd: Use SCalable Automatic REpairing with Maximal Likelihood and Bounded Changes. In *SIGMOD*. ACM.
[127] Yan Yan, Stephen Meyles, Aria Haghighi, and Dan Suciu. 2020. Entity matching in the wild: A consistent and versatile framework to unify data in industrial applications. In *SIGMOD*. 2287–2301.
[128] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *ICML*. PMLR, 5675–5684.
[129] zeorb. 2018. How do I split a TV Series into 2 tv series? https://community-imdb.sprinklr.com/conversations/data-issues-policy-discussions/how-do-i-split-a-tv-series-into-2-tv-series/5f4a79fa8815453dba940741.
[130] Aoqian Zhang, Shaoxu Song, Yu Sun, and Jianmin Wang. 2019. Learning individual models for imputation. In *ICDE*. IEEE, 160–171.
[131] 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.
[132] Dongxiang Zhang, Dongsheng Li, Long Guo, and Kian-Lee Tan. 2020. Unsupervised entity resolution with blocking and graph algorithms. *TKDE* 34, 3 (2020), 1501–1515.
[133] Wen Zhang, Bibek Paudel, Liang Wang, Jiaoyan Chen, Hai Zhu, Wei Zhang, Abraham Bernstein, and Huajun Chen. 2019. Iteratively learning embeddings and rules for knowledge graph reasoning. In *WWW*. 2366–2377.
[134] Yiliang Zhang and Qi Long. 2021. Fairness in Missing Data Imputation. *CoRR* abs/2110.12002 (2021).
[135] 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.

Received April 2023; revised July 2023; accepted September 2023