Synthesizing Privacy Preserving Entity Resolution Datasets

Xuedi Qin[1], Chengliang Chai[1*], Nan Tang[2], Jian Li[1], Yuyu Luo[1], Guoliang Li[1*], Yaoyu Zhu[1]
[1]*Department of Computer Science, Tsinghua University* [2]*QCRI, HBKU*
{qxd17@mails., ccl@mail., lijian83@mail., luoyy18@mails., liguoliang@, zyy18@mails.}tsinghua.edu.cn, ntang@hbku.edu.qa

***Abstract*—Entity resolution (ER) is a core problem in data integration. Many companies have lots of datasets where ER needs to be conducted to integrate the data. On the one hand, it is nontrivial for non-ER experts within companies to design ER solutions. On the other hand, most companies are reluctant to release their real datasets for multiple reasons (*e.g.,* privacy issues). A typical solution from the machine learning (ML) and the statistical community is to create surrogate (*a.k.a.* analogous) datasets based on the real dataset, release these surrogate datasets to the public to train ML models, such that these models trained on surrogate datasets can be either directly used or be adapted for the real dataset by the companies.**

In this paper, we study a new problem of synthesizing surrogate ER datasets using transformer models, with the goal that the ER model trained on the synthesized dataset can be used directly on the real dataset. We propose privacy preserving methods to synthesize ER datasets: we first learn the true similarity distributions of both matching and non-matching entity pairs from real dataset. We then devise algorithms that satisfy differential privacy and can synthesize fake but semantically meaningful entities, add matching and non-matching labels to these fake entity pairs, and ensure that the fake and real datasets have similar distributions. We also describe a method for entity rejection to avoid synthesizing bad fake entities that may destroy the original distributions. Extensive experiments show that ER matchers trained on real and synthetic ER datasets have very close performance on the same test sets – their *F1 scores* differ within 6% on 3 commonly used ER datasets, and their average *precision, recall* differences are less than 5%.

***Index Terms*—Data Synthesis, Entity Resolution**

I. Introduction

Entity resolution (ER) is a fundamental problem of data integration [1], [2], [3], [4]. Although there are plenty of real-world ER datasets within many companies, most companies are reluctant to release them to the public for many different reasons (*e.g.,* privacy issues). A common strategy for companies to share their datasets is through publishing surrogate datasets [5], [6], [7], [8] that are analogous to the real datasets. One practical goal is that the machine learning (ML) models trained on surrogate datasets can be either directly used or be adapted for the real dataset.

Analogously, if we can synthesize surrogate ER datasets $\mathbf{E}_{\mathsf{syn}}$ that resemble the real ER datasets $\mathbf{E}_{\mathsf{real}}$ and are purely fake, it is easier to convince the companies to release these surrogate ER datasets. This will benefit both ER researchers where they can get more data to study ER models and data owners where they can get more effective models.

Desiderata of Synthesized ER Datasets. We consider each ER dataset $\mathbf{E}$ with three parts (D, M, N), where D contains entities, and M and N are annotated labels of matching and non-matching entity pairs. Next, let's discuss the desiderata of a synthesized ER dataset $\mathbf{E}_{\mathsf{syn}} = (D_{\mathsf{syn}}, M_{\mathsf{syn}}, N_{\mathsf{syn}})$ *w.r.t.* a real one $\mathbf{E}_{\mathsf{real}} = (D_{\mathsf{real}}, M_{\mathsf{real}}, N_{\mathsf{real}})$.

1) *Indistinguishable entities:* Given an entity e, one cannot tell that whether e is from D_{real} or D_{syn}.
2) *Performance preservation:* For an ER model (or a "matcher") $\mathbf{M}$ (*e.g.,* a random forest [9] or a deep neural network [10], [11]), the matcher $\mathbf{M}_{\mathsf{real}}$ trained on $\mathbf{E}_{\mathsf{real}}$ and the matcher $\mathbf{M}_{\mathsf{syn}}$ trained on $\mathbf{E}_{\mathsf{syn}}$ should have similar test performance (*e.g., precision* and *recall*) on the same test set $\mathbf{T}$, *i.e.,* $\mathbf{M}_{\mathsf{syn}}(\mathbf{T}) \approx \mathbf{M}_{\mathsf{real}}(\mathbf{T})$.
3) *Privacy preserving:* The synthesized ER dataset $\mathbf{E}_{\mathsf{syn}}$ should not leak privacy *w.r.t.* real entities of $\mathbf{E}_{\mathsf{real}}$.

Challenges. There are two main challenges.

(C1) *Entity level similarity.* We need to synthesize meaningful entities that contain a mix of attribute types, such as numeric, categorical, and textual attributes (*e.g.,* paper title).

(C2) *Entity pair level similarity.* In order to achieve our goal $\mathbf{M}_{\mathsf{syn}}(\mathbf{T}) \approx \mathbf{M}_{\mathsf{real}}(\mathbf{T})$, we need to further ensure that the distribution between entity pairs, which models the *similarity vectors* between matching and non-matching entity pairs, should resemble that of $\mathbf{E}_{\mathsf{real}}$.

Contributions. We propose methods for synthesizing ER datasets. We first learn the distribution of $\mathbf{E}_{\mathsf{real}}$ based on the similarity vectors of matching and non-matching entity pairs, from which we then sample similarity vectors and synthesize entity pairs based on the sampled similarity vectors.

We summarize our contributions as follows:

1) We define a new problem of *synthesizing ER datasets* and propose a novel framework to solve this problem (Section II).
2) We devise algorithms for synthesizing fake entities that resemble real entities (*i.e.,* entity level), as well as entity pairs that satisfy the similarity vectors sampled from $\mathbf{E}_{\mathsf{real}}$ (Sections IV & VI).
3) We further design an entity rejection method, in order to reject the synthesized entities that either look unreal (*i.e.,*

* Guoliang Li and Chengliang Chai are the corresponding authors.

entity level) or may destroy the distribution (*i.e.,* entity pair level) that $\mathbf{E}_{\mathsf{syn}}$ should follow (Section V).

4) We devise privacy preserving methods to synthesize ER datasets so that the privacy of real datasets will not be leaked: we train transformer models differential privately (Section VI).
5) We conduct extensive experiments on synthesized ER datasets based on 3 commonly used ER datasets. The experimental results show that the matchers $\mathbf{M}_{\mathsf{syn}}$ trained on $\mathbf{E}_{\mathsf{syn}}$ and $\mathbf{M}_{\mathsf{real}}$ trained on $\mathbf{E}_{\mathsf{real}}$ have very close performance when being validated on the same test set: the differences of their *F1 scores* are within 6% (Section VII).

Novelty. There are several related works that are close to this work, but they cannot be used to synthesize surrogate ER datasets which resemble real ones in both entity and entity pair levels. (1) ZeroER [12] is designed to find the matching pairs by guessing the matching and non-matching distributions of $\mathbf{E}_{\mathsf{real}}$ without any labels being provided, and it cannot synthesize ER datasets; (2) EMBench [13], [14] synthesizes new entities by modifying the entities in $\mathbf{E}_{\mathsf{real}}$. They do not require the new entity pairs hold the same distribution with real ones and do not ensure privacy; (3) GAN based works [15], [16], [17], [18] can only synthesize one table. They train GAN models to learn the distribution of the real table, and synthesize a fake table by the generator of GAN. Although they can be used to synthesize relational tables of the ER dataset one by one, they cannot guarantee the similarity vector distribution between the synthesized tables is the same as real ones because each table of the ER dataset is synthesized independently.

II. Preliminary and Problem Definition

A. (Synthesized) ER Datasets

ER Datasets. Let A and B be two relations and $a \in A, b \in B$ denote two entities. It is known that some entities are common (*i.e.,* matches) to A and B. The set of pairs:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

is the union of two disjoint sets:

$$M = \{(a, b) \mid a = b, a \in A, b \in B\}, \text{ and}$$
$$N = \{(a, b) \mid a \neq b, a \in A, b \in B\}$$

which we call matching and non-matching sets, respectively. Let $\mathbf{E} = (A, B, M, N)$ denote an ER dataset, where every entity pair between A and B is labeled as either matching or non-matching. We also write $D = (A, B)$ and $\mathbf{E} = (D, M, N)$.

Each entity contains a number of attributes (*e.g.,* name, age, gender, marital status, address, DOB, etc). Typically, we assume that there is a one-to-one attribute correspondence (or mapped schemas) from an A-entity to a B-entity [9], [12], but the attribute names can be different (*e.g.,* gender and sex).

Example 1: Figure 1 shows a sample ER dataset. Figure 1(a) is an excerpt of DBLP relation (A-relation) and Figure 1(b) is an excerpt of ACM relation (B-relation). DBLP and ACM have the same attributes, where *id* is used to identify an entity, and the matching entities are marked in the same color. There are 2 matching pairs: (a_1, b_1) and (a_2, b_2), *i.e.,* $M = \{(a_1, b_1), (a_2, b_2)\}$, and $3 \times 3 - 2 = 7$ non-matching pairs, *i.e.,* $N = \{(a_1, b_2), (a_1, b_3), (a_2, b_1), ...\}$. □

id	title	authors	venue	year
a_1	Adaptable Query Optimization and Evaluation in Temporal Middleware	Christian S. Jensen, Richard T. Snodgrass, Giedrius Slivinskas	SIGMOD Conference	2001
a_2	Generalised Hash Teams for Join and Group-by	Donald Kossmann, Alfons Kemper, Christian Wiesner	VLDB	1999
a_3	A simple algorithm for finding frequent elements in streams and bags	Scott Shenker, Christos H. Papadimitriou, Richard M. Karp	ACM Trans. Database Syst.	2003

(a) An excerpt of DBLP.

id	title	authors	venue	year
b_1	Adaptable query optimization and evaluation in temporal middleware	Giedrius Slivinskas, Christian S. Jensen, Richard Thomas Snodgrass	International Conference on Management of Data	2001
b_2	Generalised Hash Teams for Join and Group-by	Alfons Kemper, Donald Kossmann, Christian Wiesner	Very Large Data Bases	1999
b_3	Parameterized complexity for the database theorist	Martin Grohe	ACM SIGMOD Record	2002

(b) An excerpt of ACM.

	x	entity pair	sim_title	sim_authors	sim_venue	sim_year
X^+	$\mathbf{x}_1^+$	(a_1, b_1)	1.0	0.72	0.16	1.0
	$\mathbf{x}_2^+$	(a_2, b_2)	1.0	0.86	0.04	1.0
X^-	$\mathbf{x}_1^-$	(a_1, b_2)	0.07	0.08	0.0	0.8
	$\mathbf{x}_2^-$	(a_1, b_3)	0.01	0.0	0.15	0.9
	$\mathbf{x}_3^-$	(a_2, b_1)	0.07	0.10	0.0	0.8
	...	...	...	...	...	...

(c) Matching similarity vectors X^+ and non-matching similarity vectors X^- on (a) and (b).

Fig. 1. An ER dataset with two tables: (a) DBLP and (b) ACM. (c) The matching similarity vectors X^+ and non-matching similarity vectors X^-.

Synthesized ER Datasets. A synthesized ER dataset $\mathbf{E}_{\mathsf{syn}} = (A_{\mathsf{syn}}, B_{\mathsf{syn}}, M_{\mathsf{syn}}, N_{\mathsf{syn}})$ is *just* an ER dataset, but with *everything being synthesized*, including entities in A_{syn} and B_{syn} and the matching and non-matching labels in M_{syn} and N_{syn}.

Essentially, an ER model is to learn the distributions of matching and non-matching entity pairs. So if we can synthesize a dataset $\mathbf{E}_{\mathsf{syn}}$ whose distributions of matching and non-matching pairs resemble those of $\mathbf{E}_{\mathsf{real}}$, the ER model learned from $\mathbf{E}_{\mathsf{syn}}$ will be similar to the one learned from $\mathbf{E}_{\mathsf{real}}$ [15], [17]. Hence, the distributions of matching and non-matching entity pairs of $\mathbf{E}_{\mathsf{real}}$ are crucial for synthesizing ER datasets.

B. Matching and Non-matching Distributions

A convenient way of modeling the distributions of entity pairs is to represent each entity pair as a similarity vector (*a.k.a.* feature vector) and model these similarity vectors [12], [19], [9], [20].

Similarity Vector of An Entity Pair. Let $\{C_1, C_2, ..., C_l\}$ be the aligned schema between A and B. Let $\{f_1, f_2, ..., f_l\}$ be the corresponding similarity functions. Given an entity pair (a, b), the *similarity vector* $\mathbf{x}_{(a,b)}$ of (a, b) is:

$$\mathbf{x}_{(a,b)} = (f_i(a[C_i], b[C_i]) \mid 1 \leq i \leq l)$$

Example 2: Figure 1(c) shows sample similarity vectors. The similarity functions of columns *title*, *authors*, *venue* are 3-gram jaccard similarity, and the similarity function of column *year* is $f(year_1, year_2) = 1 - \frac{|year_1 - year_2|}{\max(year) - \min(year)}$, where $\max(year)$ and $\min(year)$ are the maximum and minimum values of column $year$, and the $\max(year) - \min(year) = 10$. For example, the *venue* similarity of pair (a_1, b_1) is $3\_gram\_jaccard($"`SIGMOD Conference`", "`Internati`

onal Conference on Management of Data") = 0.16, and the *year* similarity of pair (a_1, b_1) is $1 - |2001 - 2001|/10 = 1$. Hence, $\mathbf{x}_1^+ = (1.0, 0.72, 0.16, 1.0)$ is the similarity vector of (a_1, b_1). The similarity vectors of other entity pairs are computed similarly. □

Similarity Vectors of Matching/Non-matching Entity Pairs. In $\mathbf{E} = (A, B, M, N)$, an entity pair (a, b) is either a matching pair in M, or a non-matching pair in N. So the similarity vectors of all entity pairs in $\mathbf{E}$ can be categorized as:

$$X^+ = \{\mathbf{x}_{(a,b)} \,|\, (a,b) \in M\}$$
$$X^- = \{\mathbf{x}_{(a,b)} \,|\, (a,b) \in N\}$$

where X^+ (resp. X^-) is the set of all similarity vectors for matching (resp. non-matching) pairs. For simplicity, we write $\mathbf{x}_{(a,b)}$ as $\mathbf{x}$ when it is clear from the context.

Example 3: Figure 1(c) shows two matching similarity vectors of $(a_1, b_1), (a_2, b_2)$ in X^+, and seven non-matching similarity vectors of $(a_1, b_2), (a_1, b_3), (a_2, b_1), ...$ in X^-. □

Matching and Non-matching Distributions. As observed by [12], the similarity vectors for matching pairs should look different from those of non-matching pairs. That is, if an entity pair is a match, their similarity vector should follow a matching distribution, namely the $\mathcal{M}$-distribution; otherwise, the similarity vectors of non-matching pairs should follow a different non-matching distribution, namely the $\mathcal{N}$-distribution. In fact, $\mathcal{M}$- and $\mathcal{N}$-distributions are true but unknown distributions, and X^+ and X^- are samples and are subject to these two distributions, respectively. Moreover, the $\mathcal{M}$- and $\mathcal{N}$-distributions together form the overall mixture distribution of all similarity vectors, denoted by $\mathcal{O}$-distribution.

Let the probability density function (PDF) of $\mathcal{M}/\mathcal{N}$-distribution be $p_m(x)/p_n(x)$, where x is the similarity vector variable. Let the probability of matching be $\pi = \frac{|X^+|}{|X^+|+|X^-|}$. The PDF of $\mathcal{O}$-distribution is represented as:

$$p(x) = \pi p_m(x) + (1 - \pi) p_n(x).$$

Please refer to Figure 3(c) for $\mathcal{M}$- and $\mathcal{N}$-distributions.

C. Differential Privacy

Differential privacy (DP) [21], [22] provides strong privacy guarantees for sharing information of datasets. Intuitively, an algorithm is differentially private if the attacker cannot tell the computation results of the algorithm on two adjacent datasets. Two datasets are adjacent if they only differ in one individual information. So attackers cannot identify individual information of the shared data if the algorithm for data sharing is differentially private. A randomized algorithm $\mathcal{A}$ satisfies (ϵ, δ)-differential privacy if for any computation range $\mathcal{S}$ of $\mathcal{A}$, and for any two adjacent datasets D and D', we have:

$$Pr[\mathcal{A}(D) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{A}(D') \in \mathcal{S}] + \delta \qquad (1)$$

where ϵ and δ measure the privacy level of $\mathcal{A}$: The smaller ϵ and δ are, the higher the degree of privacy preserving is.

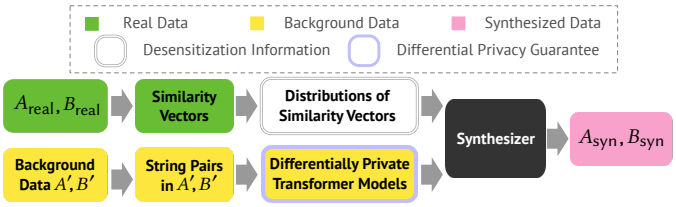


Fig. 2. How to Achieve Privacy Preserving.

D. Problem Statement

Our goal is to synthesize an ER dataset $\mathbf{E}_{\text{syn}}$ based on $\mathbf{E}_{\text{real}}$. More formally, the problem is defined as follows.

Synthesizing ER Datasets. Given the $\mathcal{O}$-distribution of a real ER dataset $\mathbf{E}_{\text{real}}$, and the desired sizes of the synthesized tables: n_a for A_{syn} and n_b for B_{syn} (by default, $n_a = |A_{\text{real}}|$ and $n_b = |B_{\text{real}}|$, where $|A_{\text{real}}|$ and $|B_{\text{real}}|$ are the size of A_{real} and B_{real}, respectivly), the problem of *synthesizing ER datasets* is to synthesize another ER dataset $\mathbf{E}_{\text{syn}}$ $(A_{\text{syn}}, B_{\text{syn}}, M_{\text{syn}}, N_{\text{syn}})$, such that $|A_{\text{syn}}| = n_a, |B_{\text{syn}}| = n_b$ (where $|A_{\text{syn}}|$ and $|B_{\text{syn}}|$ are the size of A_{syn} and B_{syn}, respectivly), and the performance difference between $\mathbf{M}_{\text{syn}}$ trained on $\mathbf{E}_{\text{syn}}$ and $\mathbf{M}_{\text{real}}$ trained on $\mathbf{E}_{\text{real}}$ on the same test set $\mathbf{T}$ is minimized:

$$\min |\text{eval}(\mathbf{M}_{\text{real}}(\mathbf{T})) - \text{eval}(\mathbf{M}_{\text{syn}}(\mathbf{T}))| \qquad (2)$$

where eval is an evaluation metric, *e.g., F1-score* or *precision*.

Recall the desiderata of synthesized ER datasets in Section I: (1) *indistinguishable entities*; (2) *performance preservation*, and (3) *privacy preserving*.

For (1), we use transformer models [23] to synthesize entities which resemble real ones.

For (2), we synthesize $\mathbf{E}_{\text{syn}}$ whose $\mathcal{O}$-distribution is similar to the $\mathcal{O}$-distribution of $\mathbf{E}_{\text{real}}$, with which to ensure that trained ER matchers have similar performance.

For (3), we will not use real data for synthesizing, but instead we use the data in the same domain but not in the active domain (*i.e.,* the same dataset), called *background data*. For example, if $\mathbf{E}_{\text{real}}$ contains names from the US, the background data could be names from Europe, which are easy to obtain. That is, when training the transformer models for ER dataset synthesis, we use the true $\mathcal{O}$-distribution from the real dataset but not the true entities. Even if the true similarity between two entities (*e.g.,* 0.8) can be attacked or inverted, no true entities will be identified because there are infinite number of entity pairs whose similarity could be 0.8. Naturally, the privacy of true entities will be preserved. Also, because the training data and the real data are from the same domain, the synthesized entities can resemble real entities. Besides, to also protect the privacy of training data, we train the transformer models differential privately by clipping and adding random noise to the gradients before gradient decent. By this way, the transformer model satisfies differential privacy [24].

Figure 2 explains how do we achieve *privacy preserving*. The input of the synthesizer contains two parts: (1) the distributions of similarity vectors computed by A_{real} and B_{real}. The distributions of similarity vectors will not leak any information of real entities because one similarity vector can corresponding

to infinite entity pairs; and (2) the transformer models trained by the string pairs in the *background data* A' and B'. The transformer models satisfy DP so the models will protect the privacy of A', B', and A', B' has no overlap with $A_{\mathsf{real}}, B_{\mathsf{real}}$, so the transformer models will also protect the privacy of $A_{\mathsf{real}}, B_{\mathsf{real}}$. In summary, neither of (1) and (2) will reveal privacy information of entities in A_{real} and B_{real} (*i.e.,* the green parts), so the synthesizer is *privacy preserving*.

III. Solution Overview

Let O_{real} (resp. O_{syn}) be the $\mathcal{O}$-distribution of $\mathbf{E}_{\mathsf{real}}$ (resp. $\mathbf{E}_{\mathsf{syn}}$). Before we propose our solution, let's discuss our goal.

Minimize the Difference between O_{real} and O_{syn}. Consider an ER matcher trained on O_{real}. If we can synthesize $\mathbf{E}_{\mathsf{syn}}$ whose O_{syn} resembles O_{real}, the matcher trained by $\mathbf{E}_{\mathsf{syn}}$ will behave similar to the matcher trained by $\mathbf{E}_{\mathsf{real}}$, so the problem of *synthesizing ER datasets* (Equation 2) can be transformed to minimize the Jensen-Shannon divergence (JSD) [25] between O_{real} with the PDF $p(x)$ and O_{syn} with the PDF $q(x)$:

$$\min \text{JSD}(p \,||\, q) = \frac{1}{2}\,\text{KL}(p \,||\, \frac{p+q}{2}) + \frac{1}{2}\,\text{KL}(q \,||\, \frac{p+q}{2}) \quad (3)$$

where $||$ is to represent the relationship between two distributions p and q, KL is the Kullback-Leibler divergence [25].

Our goal is to synthesize $\mathbf{E}_{\mathsf{syn}}$ such that O_{syn} is equal to (or close to) O_{real}. Clearly we do not want our solution to be a duplicate of $\mathbf{E}_{\mathsf{real}}$, so assume we only know the distribution O_{real} we want to match (but not the original dataset $\mathbf{E}_{\mathsf{real}}$).

Now, we discuss the tractability of our problem. Since our problem is not a decision problem, the notion of NP-completeness is not directly applicable. However, here we define a closely related decision problem, the SynER-Decision problem (which can be thought as a special case of our problem). We prove the SynER-Decision problem is NP-Complete, indicating that our problem (with only input O_{real}) is also computationally intractable. Now, we formally define the SynER-Decision problem. In this problem, we are given A_{syn}, which consists of n records (i.e., we already have one relation, and we need to synthesize the other relation B_{syn}). Both A_{syn} and B_{syn} have the same attributes "id" and "title". We are also given the $\mathcal{M}$-distribution (we only have matching set for simplicity). B_{real} should consist of only 1 record, the similarity function used here is the edit distance. The goal of the decision problem is to decide whether there is a record of B_{syn} that can satisfy the given $\mathcal{M}$-distribution exactly.

Theorem 1: *The SynER-Decision problem is NP-Complete.* □

Proof: Since the only non-id attribute is "title", each record is a string. Clearly, this problem is a decision problem and one can verify if a given string (as the record of B_{syn}) is a valid solution by computing the edit distances between this string and all strings in A_{syn} in polynomial time (hence the $\mathcal{M}$-distribution). Now, we provide a reduction from the following *central string problem* which is known to be NP-complete [26]. The central string problem is defined as follows: we are given n strings $s_1, \ldots, s_n$, and the problem is to decide whether there is a string s such that the edit distance between s and s_i is at most k for all $1 \le i \le n$. In fact, if one examines the proof of [26], one can see that the problem of determining whether there exists string s such that the edit distance between s and s_i is equal to k for all i is also NP-Complete. Given a central string problem instance, we can naturally construct a SynER-Decision instance as follows: A_{syn} consists of n strings $s_1, \ldots, s_n$. The given $\mathcal{M}$-distribution is only supported on k (i.e., with probability 1, its takes value k). Hence, synthesizing the record of B_{syn} is equivalent to finding a string s whose edit distance to s_i is exactly k for all i, hence the NP-Completeness of SynER-Decision. □

In summary, we can conclude that synthesizing $\mathbf{E}_{\mathsf{syn}}$ such that O_{syn} is equal to O_{real} is a computational intractable problem. Hence, we propose a heuristic method to synthesize $\mathbf{E}_{\mathsf{syn}}$ such that O_{syn} is similar with O_{real} in this section.

We design a framework for *synthesizing ER datasets* as shown in Figure 3. The Algorithm SERD (**S**ynthesize **ER** **D**atasets) with three steps is overviewed below, and technical details are postponed to latter sections.

S1. [Learn Distributions from $\mathbf{E}_{\mathsf{real}}$.] Given A_{real} and B_{real} (Figure 3(a)), we first compute the matching similarity vectors X^{+}_{real} and non-matching similarity vectors X^{-}_{real} of $\mathbf{E}_{\mathsf{real}}$ (Figure 3(b)). We then learn the $\mathcal{M}$- and $\mathcal{N}$-distributions from X^{+}_{real} and X^{-}_{real}, respectively (Figure 3(c)).

S2. [Synthesize $\mathbf{E}_{\mathsf{syn}}$.] We bootstrap by manually preparing one fake A-entity a, *i.e.,* $A_{\mathsf{syn}} = \{a\}$ and $B_{\mathsf{syn}} = \{\}$ (we discuss how to get an entity a later). Then, the following of this step is an iterative process. In each iteration, we sample a synthesized entity e from $A_{\mathsf{syn}} \cup B_{\mathsf{syn}}$ and a similarity vector $\mathbf{x}$ from O_{real}, which are used to synthesize a new entity e'. If $|A_{\mathsf{syn}}| = n_a$ and $|B_{\mathsf{syn}}| = n_b$, this step is complete, and we go to step **S3**.

a) S2-1. [Sample a Synthesized Entity.]: Randomly select an entity e from $A_{\mathsf{syn}} \cup B_{\mathsf{syn}}$.

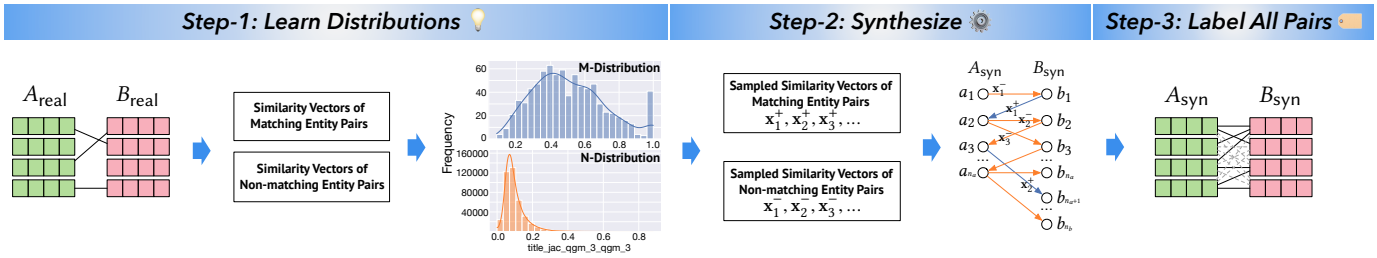
b) S2-2. [Sample a Similarity Vector.]: Sample a similarity vector $\mathbf{x}$ from $\mathcal{M}$-distribution with probability π and from $\mathcal{N}$-distribution with probability $1 - \pi$ (Figure 3(d)).

c) S2-3. [Synthesize a New Entity.]: If $\mathbf{x}$ is sampled from $\mathcal{M}$-distribution, we synthesize an entity e' that matches e; otherwise, if $\mathbf{x}$ is sampled from $\mathcal{N}$-distribution, we synthesize e' that will not match e (Figure 3(e)), where e' is synthesized based on e and $\mathbf{x}$ such that the similarity vector of e' and e is $\mathbf{x}$ (see Section IV for details).

d) S2-4. [Add Synthesized Entity to $\mathbf{E}_{\mathsf{syn}}$.]: If e is sampled from A_{syn}, we add e' to B_{syn}; otherwise (e is sampled from B_{syn}), we add e' to A_{syn}. Moreover, if $\mathbf{x}$ is sampled from $\mathcal{M}$-distribution, we add (e, e') to M_{syn}; otherwise, we add (e, e') to N_{syn}.

e) S2-5. [Loop.]: Go to step **S2**.

S3. [Label All Pairs.] As a dataset, $\mathbf{E}_{\mathsf{syn}}$ should have labels for all pairs of entities, so we should label the entity pairs which are not in $M_{\mathsf{syn}} \cup N_{\mathsf{syn}}$ as matching or non-matching



(a) The Real ER Dataset (b) Compute Similarity Vectors (c) Learn Distributions (d) Sample Similarity Vectors (e) Synthesize Entities (f) The Synthesized ER Dataset

Fig. 3. Solution overview of SERD.

based on the probabilities of their similarity vectors belonging to $\mathcal{M}$-distribution or $\mathcal{N}$-distribution (Figure 3(f)).

Remarks. (1) If A_{syn} (resp., B_{syn}) has reached the number of requirement, *i.e.,* n_a (resp., n_b), there is no need to synthesize entities for A_{syn} (resp., B_{syn}). In this case, we only sample entity e from A_{syn} (resp., B_{syn}) in step *S2-1* and synthesize entity e' for B_{syn} (resp., A_{syn}) in step *S2-3* and step *S2-4*. (2) How to synthesize a fake A-entity a in the beginning of step **S2** is discussed in Section IV-B2. (3) We also present an entity rejection technique in Section V, which rejects the synthesized entities that may destroy the $\mathcal{O}$-distribution.

Example 4: Figure 3(e) shows the step **S2** of Algorithm SERD. The circles are entities (the left and right circles are synthesized entities in A_{syn} and B_{syn} respectively) and the arrows denote sampled similarity vectors (the blue and yellow arrows denote matching and non-matching similarity vectors sampled from the $\mathcal{M}$- and $\mathcal{N}$-distributions respectively). Suppose $n_a \leq n_b$ in this case.

At first, $A_{\text{syn}} = B_{\text{syn}} = \{\}$, we synthesize a new entity a_1 for A_{syn}. Then we begin the iterations of **S2**: (1) We first sample a synthesized entity $e = a_1$ from $A_{\text{syn}} = \{a_1\}$ (*S2-1*); after that, we sample a similarity vector from O_{real}, and suppose it is $\mathbf{x}_1^-$, which means that $\mathbf{x}_1^-$ is sampled from $\mathcal{N}$-distribution (*S2-2*). We then synthesize an entity b_1 that does not match a_1 and the similarity vector of a_1 and b_1 is $\mathbf{x}_1^-$ (*S2-3*). Afterwards, we add b_1 to B_{syn}, and add (a_1, b_1) to N_{syn} (*S2-4*). After this iteration, $A_{\text{syn}} = \{a_1\}, B_{\text{syn}} = \{b_1\}$. (2) In the second iteration, we first sample $e = b_1$ from $A_{\text{syn}} \cup B_{\text{syn}} = \{a_1, b_1\}$ (*S2-1*); after that, we sample a similarity vector from O_{real}, and suppose it is $\mathbf{x}_1^+$, which means that $\mathbf{x}_1^+$ is sampled from $\mathcal{M}$-distribution (*S2-2*). We then synthesize an entity a_2 that matches b_1 and the similarity vector of b_1 and a_2 is $\mathbf{x}_1^+$ (*S2-3*). Afterwards, $A_{\text{syn}} = \{a_1, a_2\}, B_{\text{syn}} = \{b_1\}, M_{\text{syn}} = \{(a_2, b_1)\}$ (*S2-4*). (3) In the third iteration, we sample $e = a_2$ from $A_{\text{syn}} \cup B_{\text{syn}} = \{a_1, b_1, a_2\}$ and sample $\mathbf{x}_2^-$ from O_{real}, then synthesize b_2 by a_2 and $\mathbf{x}_2^-$. Afterwards, $A_{\text{syn}} = \{a_1, a_2\}, B_{\text{syn}} = \{b_1, b_2\}, N_{\text{syn}} = \{(a_1, b_1), (a_2, b_2)\}$. We repeat these steps until $|A_{\text{syn}}| = n_a$ and $|B_{\text{syn}}| = n_b$. □

IV. ER Synthesis Framework

In this section, we provide more details for our ER synthesis framework. We start by describing how to learn the $\mathcal{M}$- and $\mathcal{N}$-distributions from $\mathbf{E}_{\text{real}}$ at step **S1** (Section IV-A). We then discuss how to synthesize a new entity at step **S2** (Section IV-B). Moreover, we present how to label the entity pairs that are not yet labeled at **S3** (Section IV-C).

A. Learning $\mathcal{M}$- and $\mathcal{N}$-Distributions

Given the similarity vectors $X^+_{\text{real}}/X^-_{\text{real}}$ of matching/non-matching pairs in $M_{\text{real}}/N_{\text{real}}$, we aim to learn the $\mathcal{M}$- and $\mathcal{N}$-distributions. We follow ZeroER [12] which models the $\mathcal{O}$-distribution as a Gaussian Mixture Model (GMM). We also model the $\mathcal{M}$- and $\mathcal{N}$-distributions in $\mathcal{O}$-distribution as multivariate GMMs. The PDF of $\mathcal{M}$-distribution is $p_m(x) = \sum_{i=1}^{g} \pi_i p_i(x; \mu_i, \Sigma_i)$, where the parameter Θ includes:

1) g is the number of normal distributions;
2) π_i is proportion of the i-th normal distribution, with $\sum_{i=1}^{g} \pi_i = 1$; and
3) $\mu_i, \Sigma_i, i \in [1, g]$, where $p_i \sim N(\mu_i, \Sigma_i)$.

The $\mathcal{N}$-distribution is similar to the $\mathcal{M}$-distribution.

Next, we describe how to learn the $\mathcal{M}$-distribution (*i.e.,* Θ) by X^+_{real}, and the learning process of $\mathcal{N}$-distribution is similar.

Estimating the Parameters of $\mathcal{M}$-Distribution. First, the optimal number of normal distributions g can be derived by minimizing the Akaike information criterion (AIC) [27] of X^+_{real}. Then, for $X^+_{\text{real}} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, we want to compute the Θ that maximizes the log likelihood in Equation 4 as shown below, where the Expectation-Maximization (EM) algorithm [28] can be applied to solve the problem.

$$\log L(\Theta | X^+_{\text{real}}) = \log \prod_{i=1}^{n} \sum_{j=1}^{g} \pi_j p_j(\mathbf{x}_i | \mu_j, \Sigma_j) \tag{4}$$

The EM algorithm is an iterative process. At first, we randomly initialize μ_k, Σ_k, π_k as $\mu_k^{(0)}, \Sigma_k^{(0)}, \pi_k^{(0)}, k \in [1, g]$. Then we begin the iterations. In iteration t ($t \geq 1$), there are two steps:

1) **Expectation** (**E**-step): evaluate the possibility that $\mathbf{x}_i$ belongs to the k-th normal distribution given the parameter estimates from the last iteration (*i.e.,* iteration $t-1$):

$$\gamma_{i,k}^{(t-1)} = \frac{\pi_k^{(t-1)} p_k(\mathbf{x}_i | \mu_k^{(t-1)}, \Sigma_k^{(t-1)})}{\sum_{j=1}^{g} \pi_j^{(t-1)} p_j(\mathbf{x}_i | \mu_j^{(t-1)}, \Sigma_j^{(t-1)})} \tag{5}$$

2) **Maximization** (**M**-step): re-estimates parameters by maximizing Equation 4, then we can get the new parameter estimates for iteration t:

$$\mu_k^{(t)} = \frac{1}{\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)}} \sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} \mathbf{x}_i$$

$$\Sigma_k^{(t)} = \frac{1}{\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)}} \sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} (\mathbf{x}_i - \mu_k^{(t)})(\mathbf{x}_i - \mu_k^{(t)})^\top$$

$$\pi_k^{(t)} = \frac{1}{n} \sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} \tag{6}$$

The above iteration repeats until Equation 4 is converged.

Estimating the Parameters of $\mathcal{N}$-Distribution. The $\mathcal{N}$-distribution is estimated similarly, from X^{-}_{real}.

B. Synthesizing Entities

As discussed in Section III, Equation 2 is intractable to solve. Thus, we cannot generate all entities at once, with their pairs satisfying the O_{real}. Therefore, we propose a heuristic sampling approach that samples from O_{real} iteratively. In each iteration, we first sample a synthesized entity e and then sample a similarity vector $\mathbf{x}$ from O_{real}, based on which we synthesize another entity e' such that the similarity vector between the entity pair (e, e') is $\mathbf{x}$.

1) How to synthesize a new entity from an existing entity : Next, we discuss how to synthesize e' from a sampled entity e and a sampled similarity vector $\mathbf{x}$. Let $\mathbf{x}[i]$ denote the desired similarity of e and e' on column C_i. We synthesize e' by synthesizing the values of different columns, i.e., $e'[C_i]$, s.t. $\mathbf{x}[i] = f_i(e[C_i], e'[C_i]), i \in [1, l]$. Next, we discuss how to synthesize $e'[C_i]$ based on the column type of C_i.

Numeric. If data in column C_i is numerical, we can simply compute $e'[C_i]$ based on the similarity function f_i and $e[C_i]$.

For example, suppose that we have a numerical column *year*, and $\mathbf{x}[i] = 1 - |e[C_i] - e'[C_i]|/10$, where 10 is the difference between the maximum value and the minimum value of *year*. Then given $e[C_i] = 2008$ and $\mathbf{x}[i] = 0.8$, we can compute that $e'[C_i] = 2008 \pm (1-\mathbf{x}[i]) \times 10$, *i.e.,* $e'[C_i] = 2006$ or $e'[C_i] = 2010$, where we just sample one.

Categorical. If C_i is categorical, we consider that there are finite values in C_i, *e.g., gender* and *venue*, and thus we do not synthesize new values beyond existing ones in C_i. For ease of representation, we abuse C_i a little to denote the set of values in the column. Therefore, given $e[C_i]$ and $\mathbf{x}[i]$, we iterate $\forall e'[C_i] \in C_i$, select the one such that $\mathbf{x}[i] = f_i(e[C_i], e'[C_i])$. If we cannot find one satisfying the similarity requirement, we just return the $e'[C_i]$ so that the similarity between $e[C_i]$ and $e'[C_i]$ is the closest to $\mathbf{x}[i]$ among all values in C_i.

For example, suppose that we have a categorical column *venue*, and the similarity function f_i is the 3-gram jaccard similarity. Given $e[C_i]$="`Sigmod Conference`", $\mathbf{x}[i] = 0.23$ and C_i={"`Sigmod`", "`International Conference on Management of Data`", "`VLDB`"}, we have f_i("`Sigmod Conference`", "`Sigmod`")=0.29 that is the closest to 0.23. Hence, we return "`Sigmod Conference`" as the synthesized categorical value $e'[C_i]$.

Date. Date type has a similar synthesizing process with the numerical type.

String/Text. The similarity functions on string/text columns (*e.g.,* paper title, paper authors, and product description) are mainly string similarity functions, *e.g.,* Jaccard similarity, normalized edit distance based similarity. The challenge of synthesizing this type of data is that we need to both keep the similarity and capture the semantic. Clearly, the task of synthesizing a string from another string is Sequence-to-Sequence generation, on which transformer [23] is a natural fit (see Section VI for more details).

2) Cold start: As discussed in Section III, initially, we need one fake entity to bootstrap. In practice, manually preparing one entity is with low human cost. We can also synthesize a value for each column C_i separately. Next, we discuss how to synthesize a value based on the column type of C_i.

Numeric, Categorical, Date. We can randomly sample a value from the range or set of column C_i.

String/Text. We can pick a string which is not in the real data of column C_i from the domain knowledge of column C_i. For example, for the column paper title, we can randomly pick a paper title from Google Scholar.

Besides manually preparing an entity, we can also use the GAN [29], [15], [30], [17] model to synthesize a new entity. The GAN model includes two components: the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$. The generator $\mathcal{G}$ takes as input a random noise z, and uses deconvolution layers to transform z to a fake entity in a matrix form. The outputs of different neurons of $\mathcal{G}$ correspond to different attribute values of the entity. The discriminator $\mathcal{D}$ is a binary classifier, and the training data of $\mathcal{D}$ includes real entities labeled by 1 as well as fake entities synthesized by $\mathcal{G}$ and labeled by 0. The input of $\mathcal{D}$ is an entity in a matrix form. $\mathcal{D}$ uses convolution layers and a `sigmoid` function to transform an entity to a 0/1 label. $\mathcal{G}$ and $\mathcal{D}$ play an adversarial minimax game during the training process: $\mathcal{G}$ attempts to synthesize fake entities which will be classified as real by $\mathcal{D}$, and $\mathcal{D}$ attempts to correctly classify real and fake entities. After the GAN model is trained, we can use $\mathcal{G}$ to synthesize new fake entities that resemble real entities.

C. Labeling All Pairs

Recap that the **S2** part of algorithm SERD also synthesizes the matching or non-matching relationship between the two entities e and e', which is regarded as a label in the dataset. If $\mathbf{x}$ is from $\mathcal{M}$-distribution, e and e' are matching, *i.e.,* $(e, e') \in M_{\mathsf{syn}}$; otherwise, $(e, e') \in N_{\mathsf{syn}}$. However, there are also many entity pairs that we do not know whether they are matching or not. For each of these entity pairs, we calculate its similarity vector $\mathbf{x}$, and then compute the posterior probability $P_m(\mathbf{x})$ that $\mathbf{x}$ belongs to $\mathcal{M}$-distribution ($P_m(\mathbf{x}) = \frac{\pi p_m(\mathbf{x})}{\pi p_m(\mathbf{x}) + (1-\pi) p_n(\mathbf{x})}$) and the posterior probability $P_n(\mathbf{x})$ that $\mathbf{x}$ belongs to $\mathcal{N}$-distribution ($P_n(\mathbf{x}) = 1 - P_m(x)$). If $P_m(\mathbf{x}) \geq P_n(\mathbf{x})$, we label it as a matching pair; otherwise, we label it as a non-matching pair.

Example 5: After n_a entities in A_{syn} and the n_b entities in B_{syn} are synthesized in Figure 3(e), there are many entity pairs which are not in $M_{\mathsf{syn}} \cup N_{\mathsf{syn}}$, *i.e.,* $\{(a_1, b_2), (a_3, b_{n_b}), ...\}$. For each of these entity pairs, we compute $P_m(\mathbf{x})$ and $P_n(\mathbf{x})$, where $\mathbf{x}$ is the similarity vector of the entity pair, then we label the pair as a matching pair if $P_m(\mathbf{x}) \geq P_n(\mathbf{x})$, and non-matching one otherwise. □

V. Synthesized Entity Rejection

Our goals of synthesized datasets are: (1) *Indistinguishable entities* and (2) *Performance preservation* (Section I).

For (1), the discriminator $\mathcal{D}$ of GAN, a binary classier, is trained for distinguishing real and fake entities. Given a synthesized entity e', if $\mathcal{D}(e')$ returns true, then the synthesized e' is considered to be indistinguishable; otherwise, $\mathcal{D}(e')$ returns false, which means that this entity will be rejected.

For (2), its essential goal is to minimize the difference between O_{syn} and O_{real}, *i.e.,* Equation 3. The step **S2** (Section IV-B) in our proposed heuristic solution just guarantees that the sampled entity pairs achieve the goal well because they are sampled from O_{real}. However, O_{syn} is not only computed by the sampled pairs, but also all the other pairs across generated pairs in **S3** (Section IV-C). One hidden issue is that, all matching/non-matching pairs in the synthesized entities may not strictly follow O_{real}, which is not ideal for our synthesized ER dataset. To address this issue, we propose an entity rejection technique to check whether each synthesized entity will make the O_{syn} far from O_{real} on the fly. If so, we reject this synthesized entity and re-synthesize another entity such that the O_{syn} can be gradually closer to the O_{real}.

Putting them together, if a synthesized entity e' is rejected by either case (1) or case (2), we re-execute the process of synthesizing an entity (*i.e.,* steps *S2-1* to *S2-5* of SERD), until the synthesized entity can pass both checks. Note that e' will not always be rejected, because we can adjust the strictness of rejection by tuning some parameters for both (1) and (2).

Entity Rejection by Discriminator (Case 1). We use the discriminator $\mathcal{D}$ of the GAN model in Section IV-B2 to judge whether a synthesized entity e' resembles a real entity. We can input e' to $\mathcal{D}$, and then the `sigmoid` function of $\mathcal{D}$ can output the probability of e' being predicted to be real: if the probability is less than β, we just reject it; else, accept it, where $\beta \in [0, 1]$ is a parameter. And e' will not always be rejected by $\mathcal{D}$: if β is infinitely close to 0, $\mathcal{D}$ will predict e' as real, then e' will be accepted.

Entity Rejection by Distribution (Case 2). The idea of entity rejection by distribution is that once an entity e' is synthesized, potentially a number of new entity pairs are generated. If the current O_{syn} is moving far away from O_{real}, we reject e'. Otherwise, we add it to our synthesized dataset.

Specifically, in step **S2**, every time we sample an entity e from $A_{\mathsf{syn}} \cup B_{\mathsf{syn}}$ and a vector $\mathbf{x}$ from O_{real}, a new entity e' will be synthesized and an entity pair (e, e') will be synthesized. e and e' belong to different tables. We use $T_e(T'_e)$ to denote the table containing $e(e')$. The pair (e, e') definitely obeys O_{real} because it is sampled from the distribution, but the potential generated pairs $(e'', e'), \forall e'' \in T_e$ are not. Moreover, they may even destroy the distribution of the synthesized similarity vectors, making the O_{syn} far away from the O_{real}. In this case, we should reject e'. Next, we first introduce how to compute O_{syn} given synthesized entities in A_{syn} and B_{syn}, and then describe how to update O_{syn} every time e' is synthesized.

a) Compute O_{syn}.: We denote the similarity vectors of entity pairs in $\mathbf{E}_{\mathsf{syn}}$ as X_{syn}, which is utilized to compute O_{syn}. In the beginning, A_{syn} and B_{syn} are small, and X_{syn} can be computed efficiently:

$$X_{\mathsf{syn}} = \{\mathbf{x}_{(a,b)} \,|\, (a, b) \in A_{\mathsf{syn}} \times B_{\mathsf{syn}}\}$$

Similar to Equation 6, the computation of O_{syn} needs the labels of each pair. Unlike X^+_{real} and X^-_{real}, other entity pairs in X_{syn} do not have labels, so we assign labels to them based on O_{real}. $X^+_{\mathsf{syn}}(X^-_{\mathsf{syn}})$ denotes the vectors of matching(non-matching) entity pairs in X_{syn}, which is computed as follows,

$$\begin{aligned} X^+_{\mathsf{syn}} &= \{\mathbf{x} \,|\, P_m(\mathbf{x}) \geq P_n(\mathbf{x}), \mathbf{x} \in X_{\mathsf{syn}}\} \\ X^-_{\mathsf{syn}} &= \{\mathbf{x} \,|\, P_n(\mathbf{x}) > P_m(\mathbf{x}), \mathbf{x} \in X_{\mathsf{syn}}\} \end{aligned} \tag{7}$$

Then we compute π as $\frac{|X^+_{\mathsf{syn}}|}{|X_{\mathsf{syn}}|}$, and $\mathcal{M}/\mathcal{N}$-distributions following Equation 6, and thus O_{syn} can be obtained.

b) Update O_{syn}.: Once e' is synthesized, more entity pairs between e' and entities in T_e are also synthesized. We denote the set ΔX_{syn} as vectors of these pairs,

$$\Delta X_{\mathsf{syn}} = \{\mathbf{x}_{(a,b)} \,|\, a = e', b \in T_e\}$$

To update O_{syn}, we first compute $\Delta X^+_{\mathsf{syn}}$ similar to Equation 7,

$$\Delta X^+_{\mathsf{syn}} = \{\hat{\mathbf{x}} \,|\, P_m(\hat{\mathbf{x}}) \geq P_n(\hat{\mathbf{x}}), \hat{\mathbf{x}} \in \Delta X_{\mathsf{syn}}\}$$

$\Delta X^-_{\mathsf{syn}}$ can also be computed similarly.

Next we describe how to update π and $\mathcal{M}/\mathcal{N}$-distributions of O_{syn}. Obviously, $\pi = \frac{|X^+_{\mathsf{syn}}|+|\Delta X^+_{\mathsf{syn}}|}{|X_{\mathsf{syn}}|+|\Delta X_{\mathsf{syn}}|}$; Then we calculate the new $\mathcal{M}$-distribution of O_{syn} when $\Delta X^+_{\mathsf{syn}}$ is added to X^+_{syn}. By following Equation 6, we can incrementally compute the updated $\hat{\mu}_k^{(t)}, \hat{\Sigma}_k^{(t)}, \hat{\pi}_k^{(t)}$ by adding the corresponding part of $\hat{\mathbf{x}}$ in $\Delta X^+_{\mathsf{syn}}$ to Equation 6. And at first, we compute $\hat{\gamma}_{i,k}^{(t-1)}$ for $\hat{\mathbf{x}}$ in $\Delta X^+_{\mathsf{syn}}$:

$$\hat{\gamma}_{i,k}^{(t-1)} = \frac{\pi_k^{(t-1)} p_k(\hat{\mathbf{x}}_i | \mu_k^{(t-1)}, \Sigma_k^{(t-1)})}{\sum_{j=1}^{g} \pi_j^{(t-1)} p_j(\hat{\mathbf{x}}_i | \mu_j^{(t-1)}, \Sigma_j^{(t-1)})} \tag{8}$$

Then we compute the updated $\hat{\mu}_k^{(t)}, \hat{\Sigma}_k^{(t)}, \hat{\pi}_k^{(t)}$ (suppose $|X^+_{\mathsf{syn}}| = n$, and $|\Delta X^+_{\mathsf{syn}}| = \hat{n}$):

$$\begin{aligned} \hat{\mu}_k^{(t)} &= \frac{1}{\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} + \sum_{i=1}^{\hat{n}} \hat{\gamma}_{i,k}^{(t-1)}} \left(\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} \mathbf{x}_i + \sum_{i=1}^{\hat{n}} \hat{\gamma}_{i,k}^{(t-1)} \hat{\mathbf{x}}_i \right) \\ \hat{\Sigma}_k^{(t)} &= \frac{\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} (\mathbf{x}_i - \hat{\mu}_k^{(t)})(\mathbf{x}_i - \hat{\mu}_k^{(t)})^\top + \sum_{i=1}^{\hat{n}} \hat{\gamma}_{i,k}^{(t-1)} (\hat{\mathbf{x}}_i - \hat{\mu}_k^{(t)})(\hat{\mathbf{x}}_i - \hat{\mu}_k^{(t)})^\top}{\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} + \sum_{i=1}^{\hat{n}} \hat{\gamma}_{i,k}^{(t-1)}} \\ \hat{\pi}_k^{(t)} &= \frac{1}{n + \hat{n}} \left(\sum_{i=1}^{n} \gamma_{i,k}^{(t-1)} + \sum_{i=1}^{\hat{n}} \hat{\gamma}_{i,k}^{(t-1)} \right) \end{aligned} \tag{9}$$

In this way, we do not need to re-initialize the parameters of the O_{syn} to random numbers and compute the parameters of O_{syn} iteratively, which is very inefficient because we have to compute the similarity vectors of all synthesized entity pairs in Equation 6. Instead, we initially set the parameters of O_{syn} as the parameters before $\Delta X^+_{\mathsf{syn}}$ is added to X^+_{syn}, and update

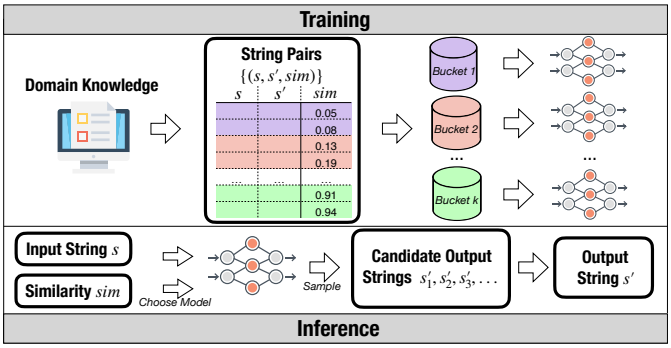


Fig. 4. Training and inference for string synthesis.

the parameters by Equation 9, so we can get the updated parameters of O_{syn} quickly. And similarly, we can compute the new $\mathcal{N}$-distribution of O_{syn}, and then we can get the updated O_{syn}, denoted by O'_{syn}.

c) Entity Rejection by Distribution.: In this part, we discuss that given O'_{syn}, whether we reject the newly synthesized entity e'. The intuition is that we should reject e' if ΔX_{syn} makes O_{syn} far away from O_{real}, *i.e.,* the JSD between O'_{syn} and O_{real} is much bigger than the JSD between O_{syn} and O_{real}:

$$\text{JSD}(O'_{\text{syn}}, O_{\text{real}}) > \alpha \, \text{JSD}(O_{\text{syn}}, O_{\text{real}}) \tag{10}$$

where $\alpha > 0$ is a parameter, and $\alpha = 1$ by default. Therefore, every time we synthesize an entity e', compute ΔX_{syn}, recompute O_{syn}, and then check whether Equation 10 holds. If yes, we reject e'; otherwise, we add e' to T'_e.

Remarks. (1) The calculation of ΔX_{syn} is time-consuming when T_e is getting larger. Hence, we can sample t entities from T_e, and then calculate the similarity vectors between e' and these t sampled entities as ΔX_{syn}. (2) e' will not always be rejected by Equation 10 because we can tune α: if α is close to $+\infty$, Equation 10 does not hold, then e' will be accepted.

VI. Synthesizing Textual Values

Recap that we want to synthesize an entity e' from a sampled entity e and a sampled similarity vector $\mathbf{x}$. For a string/text column C_i, we should synthesize a string $e'[C_i]$ such that $\mathbf{x}[i] = f_i(e[C_i], e'[C_i])$. Furthermore, $e'[C_i]$ should be semantically informative so that one cannot tell that it is a synthesized value. For example, suppose $e[C_i]$ is "`Forest Family Restaurant`", the similarity function is 3-gram jaccard similarity, and $\mathbf{x}[i]$ is 0.73, we can synthesize $e'[C_i]$ as "`De's Forest Family Restaurant`", whose similarity with $e[C_i]$ is 0.71. And for ease of presentation, we use s, s', sim, f to replace $e[C_i], e'[C_i], \mathbf{x}[i], f_i$, and then our problem can be represented as: given a string s, a similarity function f and a similarity score sim, we aim to synthesize another string s' such that $sim = f(s, s')$.

Our Solution. We train a deep learning (DL) model to solve this problem by taking as input s and sim and outputting s', *i.e.,* learning a mapping function $s' = \mathcal{F}(s, sim)$. The DL model is trained by a large number of string pairs associated with their similarity scores. Due to the powerful learning capacity of DL, we can synthesize strings that conform to the similarity requirements and have semantic meanings.

Algorithm 1: Training Transformer Model $\mathbf{M}_i$ Differential Privately

Input: Training data: string pairs $\{(s, s')\}$ whose similarity scores fall in $\mathbf{I}_i$, noise scale σ, gradient norm bound V, learning rate η

Output: Differentially Private Transformer Model $\mathbf{M}_i$

initialize the parameters θ of $\mathbf{M}_i$;
for *number of training iterations* **do**
  sample a minibatch of training examples $\{(s_1, s'_1), (s_2, s'_2), ..., (s_J, s'_J)\}$;
  $J \leftarrow$ the size of minibatch;
  $\mathcal{L} \leftarrow \mathbf{M}_i.\texttt{forward}(\{(s_1, s'_1), (s_2, s'_2), ..., (s_J, s'_J)\})$;
  for $j \leftarrow 1$ **to** J **do**
    $g(s_j, s'_j) \leftarrow \nabla_\theta \mathcal{L}(\theta, s_j, s'_j)$;
    $\overline{g}(s_j, s'_j) \leftarrow g(s_j, s'_j) / \max(1, \|g(s_j, s'_j)\|_2 / V)$;
  $\widetilde{g} \leftarrow (\sum_j \overline{g}(s_j, s'_j) + \mathcal{N}(0, \sigma^2 V^2 \mathbf{I}))/J$;
  $\theta \leftarrow \theta - \eta \widetilde{g}$;
return $\mathbf{M}_i$;

We formulate this task as a Sequence-to-Sequence (Seq2Seq) task which is commonly used in machine translation. There are many deep learning models [31], [23], [32] for Seq2Seq task. Among these models, transformer [23] has attracted lots of attention and achieved state-of-the-art results in many Seq2Seq tasks [23], [33], [34]. So we use transformer for our task. In Seq2Seq task, there are usually an encoder and a decoder. The typical transformer model encodes a string s as a hidden vector and decodes it to another string s'. In our problem, we have an additional input sim.

To solve this problem, we propose to train multiple transformer models for different similarity buckets. For $sim \in [0, 1]$, we split the interval $[0, 1]$ to k disjoint and successive intervals (buckets) $\mathbf{I}_1, \mathbf{I}_2, ..., \mathbf{I}_k$. We then train k models $\mathbf{M}_i, 0 \leq i \leq k$, where $\mathbf{M}_i$ is for pairs whose similarity scores fall in the interval $\mathbf{I}_i$. The training data for model $\mathbf{M}_i$ is the string pairs in the *background data* of column C_i whose similarity scores are in $\mathbf{I}_i$. Hopefully $\mathbf{M}_i$ can synthesize a string s' whose similarity score with s falls in the interval $\mathbf{I}_i$ when the input is s. The intuition behind this idea is that: two strings (*e.g.,* s and s') can usually be converted to each other by some underlying rules (*e.g.,* exchange the name order of *authors*). The rules of high similarities (*e.g.,* > 0.9) should be different from that of low similarities (*e.g.,* < 0.1), and the similar similarities may share similar rules. Since each model does not need sim as input, the challenge of the above method can be resolved. Moreover, $\mathbf{M}_i$ can learn the underlying rules for similarity $sim \in \mathbf{I}_i$, apply them to s, and finally synthesize s' whose similarity with s is close to sim.

Figure 4 shows the training (top) and inference processes (bottom) for string synthesis, which will be discussed next.

Training. Recall that we use background data to train the model for the third desiderata of synthesized ER datasets, *i.e., privacy preserving*. For a textual/text column C_i, we first crawl some strings which belong to the domain of C_i from domain knowledge. For example, for the column paper title, we can crawl some paper titles (*i.e.,* strings) from the website. Then

domain	input string s	sim	output string s'	sim'
authors (DBLP-ACM)	Jennifer Bernstein, Meikel Stonebraker, Guojing Lin	0.55	M. Stonebraker, G. Lin, Jennifer Bernstein	0.56
name (Restaurant)	Forest Family Restaurant	0.73	De's Forest Family Restaurant	0.71
address (Restaurant)	6th street around broadway	0.4	6th street between columbus avenue and broadway	0.4
title (Walmart-Amazon)	Asus 15.6 Laptop Intel Atom 2gb Memory 32gb Flash	0.13	Lenovo Thinkpad 15.6 Laptop	0.15
Sone_Name (iTunes-Amazon)	I'll Be Home For The Holiday	0.09	I'll Think Of You When Raining	0.11

TABLE I
EXAMPLES OF SYNTHESIZED STRINGS.

we enumerate the strings in pairs (*i.e.,* $\{(s, s')\}$), calculate the similarities of these string pairs, and divide them into buckets by their similarities. Finally, we train models for different similarity intervals (buckets).

Algorithm 1 shows how to train transformer model $\mathbf{M}_i$ differential privately. The input includes the training data of $\mathbf{M}_i$: the string pairs $\{(s, s')\}$ in the *background data* whose similarity scores are in $\mathbf{I}_i$. We first initialize the parameter θ of $\mathbf{M}_i$ (line 1), then iteratively train $\mathbf{M}_i$. In each iteration, we first sample J string pairs $\{(s_1, s'_1), (s_2, s'_2), ..., (s_J, s'_J)\}$ from the training data as a minibatch (line 3-4), then call the forward function of $\mathbf{M}_i$ and compute the loss $\mathcal{L}$ (line 5). Then compute the gradient $g(s_j, s'_j)$ for each string pair (s_j, s'_j) in the sampled minibatch (line 7), clip $g(s_j, s'_j)$ by L_2 norm with threshold V (line 8), add Gaussian noises to the clipped gradients (line 9), and update the parameter θ of $\mathbf{M}_i$ by gradient descent (line 10). After all training iterations are finished, return the trained differentially private $\mathbf{M}_i$.

Inference. After multiple transformer models are trained, for given s and sim, we first check which interval that sim falls in. Suppose $sim \in \mathbf{I}_i$, we input s to the model $\mathbf{M}_i$, and then we can get several different candidate output strings (denoted as $s'_1, s'_2, s'_3, ...$) due to the sampling process when the decoder synthesizes a token. Then we compute the similarity of s and $s'_1, s'_2, s'_3, ...$, return the string whose similarity with s is the closest to sim as s'.

Example 6: Table I shows some examples of synthesized strings for different domains, where s is the input string, sim is the input similarity, s' is the synthesized string, and sim' is the similarity of s and s'. For example, for the second row of Table I, the input of transformer model is "`Forest Family Restaurant`" and $sim = 0.73$, and the output is "`De's Forest Family Restaurant`", where $sim' = 3\_gram\_jaccard($"`Forest Family Restaurant`", "`De's Forest Family Restaurant`"$) = 0.71$. We can see that sim' is very close to sim, and the synthesized s' captures the semantic information. □

VII. EXPERIMENT

Datasets. We use 4 real-world datasets which are widely used by existing entity resolution works [19], [9], [10]. Table II shows the statistics of the 4 datasets used in our experiment, where $|A_{\mathsf{real}}|$ and $|B_{\mathsf{real}}|$ are the sizes of the two tables, $|M_{\mathsf{real}}|$ is the number of matching pairs in this dataset, and #-Col is the number of columns. (1) DBLP-ACM is a dataset of research papers. There are two relation tables: DBLP (2616 tuples) and ACM (2294 tuples). The dataset has 2224 matching entity pairs. It has 2 textual attributes: *title*, *authors*; 1 categorical attribute: *venue*; and 1 numeric attribute: *year*. (2) Restaurant is a restaurant dataset with one table which contains 864 entities. And for the case that $\mathbf{E}_{\mathsf{real}}$ only contains one table, we treat this table as both A_{real} and B_{real}, and there are 112 matching pairs except for the entity pair that matches itself. The dataset has 2 textual attributes: *name*, *address*; and 2 categorical attributes: *city*, *flavor*. (3) Walmart-Amazon is a dataset about electronic product. There are two relation tables: Walmart (2554 tuples) and Amazon (22074 tuples). The dataset has 1154 matching entity pairs. It has 3 textual attributes: *modelno*, *title*, *descr*; 1 categorical attribute: *brand*; and 1 numeric attribute: *price*. (4) iTunes-Amazon is a dataset about music albums. There are two relation tables: iTunes (6907 tuples) and Amazon (55922 tuples). The dataset has 132 matching entity pairs. It has 5 textual attributes: *song_name*, *artist_name*, *album_name*, *genre*, *copyright*; 1 numeric attribute: *price*; 2 date attributes: *time*, *released*.

Comparisons. We test 3 methods: (1) SERD; (2) SERD- is SERD but without entity rejection to show the necessity of entity rejection; and (3) EMBench [13], [14] synthesizes fake entities by modifying (*e.g.,* abbreviation, misspelling, synonyms, etc.) real entities in $\mathbf{E}_{\mathsf{real}}$, and two synthesized entities are matching (resp., non-matching) if their corresponding real entities are matching (resp., non-matching).

Settings. For categorical and textual columns, we use the 3-gram jaccard similarity to calculate their similarities; and for a numeric column C, we use $1 - \frac{|c_1 - c_2|}{\max(C) - \min(C)}$ to calculate the similarities of c_1 and c_2, where c_1, c_2 are two values on column C, and $\max(C), \min(C)$ are the maximum and minimum values on column C. We bootstrap SERD and SERD- by synthesizing the first entity automatically using the GAN model without any human cost, and we use the Daisy repository (https://github.com/ruclty/Daisy) of Fan et al. [15] to train the GAN model. This GAN model is also used to reject entities. We set the α of Equation 10 as 1, the β of the discriminator as 0.6, the number of candidate output strings (s') in Section VI as 10, and the number of similarity intervals as 10 in the experiment. We use the typical transformer model from the Attention is All You Need paper [23]. The token of the transformer is character. The input dimension is the size of the vocabulary (*i.e.,* the distinct number of characters) and the hidden dimension of the embedding layer is 256. The encoder (resp., decoder) contains 3 encoder (resp., decoder) layers and the multi head attention layer has 8 heads. The dropout rate of transformer is 0.1.

https://dbs.uni-leipzig.de/file/DBLP-ACM.zip

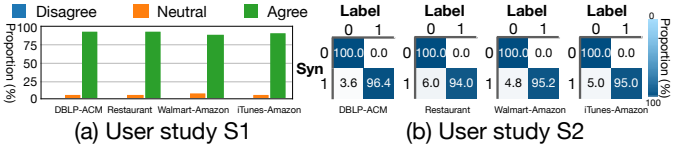
http://www.cs.utexas.edu/users/ml/riddle/data/restaurant.tar.gz

http://pages.cs.wisc.edu/~anhai/data/corleone_data/products/walmart.csv

https://pages.cs.wisc.edu/~anhai/data1/deepmatcher_data/Structured/iTunes-Amazon/

Dataset	Domain	$\|A_{\text{real}}\|$	$\|B_{\text{real}}\|$	#-Col	$\|M_{\text{real}}\|$
DBLP-ACM	scholar	2616	2294	4	2224
Restaurant	restaurant	864	864	4	112
Walmart-Amazon	electronics	2554	22074	5	1154
iTunes-Amazon	music	6907	55922	8	132

TABLE II
STATISTICS OF DATASETS.



(a) User study S1 (b) User study S2

Fig. 5. User study.

Environment. All experiments are conducted on a MacBook Pro with 16 GB 2667 MHz RAM and 2.3 GHz Intel Core i9 CPU, running OS X Version 11.1.

Exp-1: User Study

In this section, we want to show that (S1) users cannot tell that whether a synthesized entity is from $\mathbf{E}_{\text{real}}$ or $\mathbf{E}_{\text{syn}}$; (S2) the synthesized matching entity pairs are really matching and the synthesized non-matching entity pairs are really non-matching. We ask humans to answer questions for user studies S1 and S2 and verify whether our synthesized ER datasets are good.

Questions. For user study S1, we sample 500 entities from each synthesized dataset, and for each entity, ask the participants the question Q_1: *"please choose whether the entity is a real one"* with three choices {disagree, neutral, agree}. For user study S2, we sample 500 synthesized matching entity pairs and 500 synthesized non-matching entity pairs from dataset DBLP-ACM, then mix these 1000 sampled entity pairs, and for each entity pair, ask the question Q_2: *"please choose whether the entity pair is matching or non-matching"* with two choices {matching, non-matching}. We sample 100, 500, 100 matching and 100, 500, 100 non-matching entity pairs from dataset Restaurant, Walmart-Amazon, iTunes-Amazon respectively.

Participants. We employ 288 crowdsourcing workers from a crowdsourcing platform Appen (https://appen.com). The Human Intelligence Task (HIT) approval ratings of these workers are bigger than 90%. For question Q_1 (resp., Q_2), we ask 5 (resp., 3) workers to answer and aggregate their answers by majority voting.

User Study S1. Figure 5(a) shows the proportion of different answers among all questions for the 4 datasets. About 90% of synthesized entities resemble real entities (*i.e.,* get the answer Agree), and less than 4% of the synthesized entities do not resemble real entities (*i.e.,* get the answer Disagree). The experiment results show that our method can synthesize fake entities that resemble real entities because our transformer models can synthesize semantically informative strings and the discriminator of the GAN model rejects these synthetic entities which do not resemble real entities.

User Study S2. Figure 5(b) shows the proportion of different answers for synthesized matching and non-matching entity pairs. We show a matrix for each dataset, where the row is the label of the synthesized entity pair and the column is the label by workers, and the cell value is the proportion of entity pairs whose synthetic labels are the row value and the labels by users are the column value. For example, for DBLP-ACM, there are $500 \times 96.4\% = 482$ synthesized matching entity pairs labeled as matching. For the synthesized non-matching entity pairs, workers all label them as non-matching, because the synthesized non-matching entity pairs are usually not similar and the workers will not label them as matching. For the synthesized matching entity pairs, there are $\geq 94\%$ of them labeled as matching, because the transformer models can synthesize very similar strings for matching entity pair synthesis, and then the workers will label them as matching.

Exp-2: Model Evaluation

In this section, we want to show that $\mathbf{M}_{\text{syn}}$ and $\mathbf{M}_{\text{real}}$ have similar performance on the same test set, and $\mathbf{M}_{\text{syn}}$ can be used directly on the real test set. Basically, the ML models for ER can be categorized into two types: traditional ML models (*e.g.,* random forest, SVM, etc.) and deep learning models. To illustrate the applicability of our method, we test both models. For the traditional ML models, we evaluate the performance of the models trained by the Magellan system [9], [35]; for the deep learning models, we evaluate the performance of the models trained by the Deepmatcher system [10].

Metrics. To evaluate the performance of ML model trained by either real or synthesized ER dataset, we use the model to test on a test set and report the model performance by *precision*, *recall*, and *F1 score*. The closer the performance of the model trained by the synthesized ER dataset is to the performance of the model trained by the real ER dataset, the better the synthesized ER dataset is. Suppose TN is the number of entity pairs that are actually non-matching and predicted as non-matching, FP is the number of entity pairs that are actually non-matching but predicted as matching, FN is the number of entity pairs that are actually matching but predicted as non-matching and TP is the number of entity pairs that are actually matching and predicted as matching, then *precision*$= \frac{TP}{TP+FP}$, *recall*$= \frac{TP}{TP+FN}$, *F1 score*$= \frac{2\times precision\times recall}{precision+recall}$.

We synthesize ER datasets $\mathbf{E}_{\text{syn}}$ which are the same sizes as the real ER datasets by different methods (SERD and SERD-) for the 4 datasets. To test ML model (both Magellan and Deepmatcher) performance, we first split $\mathbf{E}_{\text{real}}$ to training set and test set $\mathbf{T}$, then train model $\mathbf{M}_{\text{real}}$ by the training set of $\mathbf{E}_{\text{real}}$ and train model $\mathbf{M}_{\text{syn}}$ by $\mathbf{E}_{\text{syn}}$, finally, test the performance of $\mathbf{M}_{\text{real}}$ and $\mathbf{M}_{\text{syn}}$ on $\mathbf{T}$ respectively.

Magellan Model. Figure 6 shows the performances (including *precision*, *recall*, *F1 score*) of the Magellan models trained by $\mathbf{E}_{\text{real}}$ and $\mathbf{E}_{\text{syn}}$ for different datasets. The *F1 score* differences between SERD and Real are 4.71%, 4.49%, 3.52%, 3.57% for dataset DBLP-ACM, Restaurant, Walmart-Amazon, iTunes-Amazon respectively, with an average of 4.07%, and the average *precision, recall* differences are 4.46%, 4.70%. The average *F1 score*, *precision, recall* differences between SERD- and Real are 39.88%, 46.89%, 27.09% respectively, and the average *F1 score*, *precision, recall* differences between EMBench and Real are 31.48%, 26.56%, 30.89% respectively.

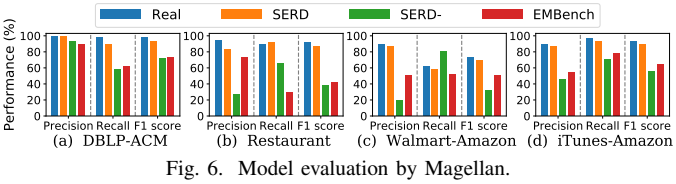


Fig. 6. Model evaluation by Magellan.

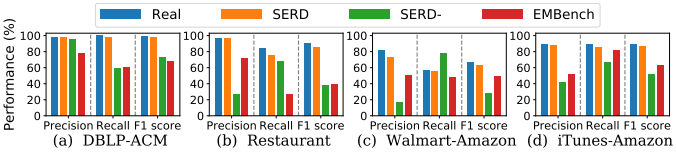


Fig. 7. Model evaluation by Deepmatcher.

Deepmatcher Model. Figure 7 shows the performances of the Deepmatcher models trained by $\mathbf{E}_{\text{real}}$ and $\mathbf{E}_{\text{syn}}$ for different datasets. The *F1 score* differences between SERD and Real are 0.83%, 5.01%, 4.11%, 2.10% for dataset DBLP-ACM, Restaurant, Walmart-Amazon, iTunes-Amazon respectively, with an average of 3.01%, and the average *precision, recall* differences are 2.24%, 3.84%. The average *F1 score*, *precision, recall* differences between SERD- and Real are 38.24%, 45.65%, 24.88% respectively, and the average *F1 score*, *precision, recall* differences between EMBench and Real are 31.33%, 28.29%, 28.31% respectively.

The experiment results confirm our point: $\mathbf{M}_{\text{real}}$ and $\mathbf{M}_{\text{syn}}$ have similar performance on a same real test set: their *F1 score* differences are less than 6%, and their average *precision, recall* differences are less than 5%, 5% respectively. SERD performs much better than SERD-, because there is no entity rejection in SERD-, and every synthesized entity will be accepted. But as illustrated in Section V, some synthesized entities will destroy the original distribution, making the O_{syn} far from the O_{real}, so the models of SERD- and Real are very different. The ML model performance differences of SERD and SERD- show that the entity rejection technique works and the accepted synthesized entities follow the O_{real}. EMBench also has big performance difference with Real, because EMBench does not require the synthesized entities have similar distributions with the real entities.

Exp-3: Data Evaluation

In this section, we want to show that $\mathbf{E}_{\text{syn}}$ and $\mathbf{E}_{\text{real}}$ have similar data characteristics: the same model tested on $\mathbf{E}_{\text{syn}}$ and $\mathbf{E}_{\text{real}}$ have similar performance. We synthesize ER datasets $\mathbf{E}_{\text{syn}}$ which are the same sizes of $\mathbf{E}_{\text{syn}}$ by SERD and SERD- for the 4 datasets. We evaluate $\mathbf{M}_{\text{real}}$ on the test set of $\mathbf{E}_{\text{real}}$ (denoted by $\mathbf{T}_{\text{real}}$) and the test set of $\mathbf{E}_{\text{syn}}$ (denoted by $\mathbf{T}_{\text{syn}}$), where $\mathbf{T}_{\text{syn}}$ is the same size as $\mathbf{T}_{\text{real}}$ and sampled from $\mathbf{E}_{\text{syn}}$.

Magellan Model. Figure 8 shows the performance differences of the Magellan models trained by $\mathbf{E}_{\text{real}}$ and tested on $\mathbf{T}_{\text{real}}$ and $\mathbf{T}_{\text{syn}}$. The *F1 score* differences between SERD and Real are 4.77%, 4.31%, 3.91%, 3.20% for dataset DBLP-ACM, Restaurant, Walmart-Amazon, iTunes-Amazon respectively, with an average of 4.05%, and the average *precision, recall* differences are 4.99%, 4.65%. The average *F1 score*, *precision, recall* differences between SERD- and Real are 15.40%, 13.84%, 17.55% respectively, and the average *F1 score*, *precision, recall* differences between EMBench and Real are 23.17%, 17.47%, 23.49% respectively.

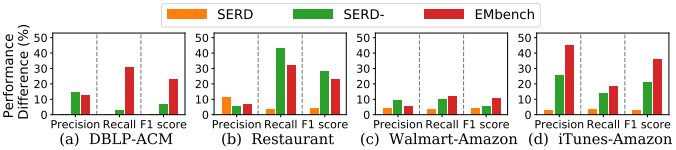


Fig. 8. Data evaluation by Magellan.

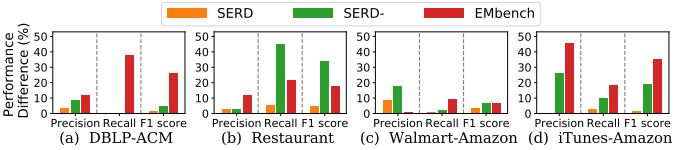


Fig. 9. Data evaluation by Deepmatcher.

Deepmatcher Model. Figure 9 shows the performance differences of the Deepmatcher models trained by $\mathbf{E}_{\text{real}}$ and tested on $\mathbf{T}_{\text{real}}$ and $\mathbf{T}_{\text{syn}}$. The *F1 score* differences between SERD and Real are 1.68%, 4.53%, 3.79%, 1.62% for dataset DBLP-ACM, Restaurant, Walmart-Amazon, iTunes-Amazon respectively, with an average of 2.90%, and the average *precision, recall* differences are 3.75%, 2.42%; The average *F1 score*, *precision, recall* differences between SERD- and Real are 16.23%, 13.84%, 14.44% respectively, and the average *F1 score*, *precision, recall* differences between EMBench and Real are 21.71%, 17.47%, 21.87% respectively.

The experiment results confirm our point: the same model test on $\mathbf{E}_{\text{syn}}$ and $\mathbf{E}_{\text{real}}$ have similar performance: the *F1 score* differences between them are less than 5%, and the average *precision, recall* differences are less than 6%, 5% respectively. For SERD, the distributions of all pairs in $\mathbf{E}_{\text{real}}$ and $\mathbf{E}_{\text{syn}}$ are similar, so the same model test on $\mathbf{E}_{\text{real}}$ and $\mathbf{E}_{\text{syn}}$ behave similarly. But for SERD-, the O_{syn} is getting far away from O_{real} without entity rejection. For EMBench, there is no guarantee that O_{syn} will be similar with O_{real}.

Exp-4: Privacy Evaluation

In this section, we show that the synthetic data can well protect privacy, to a large extent. We use two widely used metrics in existing works [36], [37], [38] for privacy evaluation.

Metrics. (1) *Hitting Rate* measures how many real entities in $\mathbf{E}_{\text{real}}$ are *similar* to a synthesized entity in $\mathbf{E}_{\text{syn}}$. Two entities are *similar* if their categorical values are the same and the similarities of their numeric/date/textual values are bigger than a threshold (we set the threshold as 0.9 in the experiment). For each synthesized entity, we compute the proportion of entities in $\mathbf{E}_{\text{real}}$ that are *similar* to the synthesized entity, *i.e., Hitting Rate*, and report the average *Hitting Rate* of all synthesized entities. (2) *Distance to the closest record (DCR)* measures whether a synthetic entity is weak from re-identification attacks [36], [37]. For a real entity in $\mathbf{E}_{\text{real}}$, we find a synthesized entity in $\mathbf{E}_{\text{syn}}$ which has the closest distance with the real entity and report the average closest distances of all entities in $\mathbf{E}_{\text{real}}$. The distance between two entities is one minus their similarity. The higher the DCR is, the better the privacy preservation is.

Table III shows the *Hitting Rate* and *DCR* of synthesized ER datasets for different datasets and algorithms. (1) By *Hitting Rate*, we can know that there are averagely $0.004\% \times (2616 + 2294) = 0.196, 0.012\% \times 864 = 0.104, 0.002\% \times (2554 + 22074) = 0.493, 0.001\% \times (6907 + 55922) = 0.628$ entities in

$\mathbf{E}_{\mathsf{real}}$ are *similar* with a synthesized entity for dataset DBLP-ACM, Restaurant, Walmart-Amazon, iTunes-Amazon respectively, which explains that the synthesized entities are very different from the real entities. Because we use *background data* as training data and the transformer models satisfy differential privacy. (2) The DCRs of SERD are larger compared to the DCRs in [10]: most of the DCRs in [10] are between 0.1 and 0.2 and the entities in [10] are synthesized by a state-of-the-art statistical data synthesis method PrivBayes [39] which has a theoretical guarantee on differential privacy [22]. Because most similarities of strings between the synthesized entities and real entities are small, and this leads to the *DCR* of ours larger than [10] (there are mainly categorical and numeric columns in [10]). Hence, re-identification attacks can not recover real entities by synthesized entities. (3) The *Hitting Rate* of EMBench is much larger than SERD and the *DCR* of EMBench is much smaller than SERD, which means that EMBench will reveal some privacy information of real entities. For example, there are averagely $0.248\% \times (6907 + 55922) = 155.8$ entities in $\mathbf{E}_{\mathsf{real}}$ are *similar* with a synthesized entity in iTunes-Amazon. EMBench synthesizes entities by modifying real entities, so the synthesized entities are similar with real entities. (4) The *Hitting Rate* and *DCR* of SERD and SERD- are similar, which illustrates that entity rejection does not affect the degree of privacy preservation.

Dataset	Hitting Rate (%)			DCR		
	SERD	SERD-	EMBench	SERD	SERD-	EMBench
DBLP-ACM	0.004	0.004	0.126	0.452	0.449	0.386
Restaurant	0.012	0.013	0.145	0.576	0.572	0.422
Walmart-Amazon	0.002	0.002	0.152	0.579	0.581	0.269
iTunes-Amazon	0.001	0.001	0.248	0.571	0.563	0.215

TABLE III
PRIVACY EVALUATION WITH $(\epsilon = 1, \delta = 10^{-5})$-DP.

Exp-5: Efficiency Evaluation

In this section, we want to show that SERD can synthesize ER datasets in a reasonable time. Table IV shows the time for synthesizing different ER datasets. The offline time is the time to train the transformer models for all textual columns and the GAN model, and the online time is the time to synthesize ER datasets. The offline time is proportional to the number of textual columns: the more textual columns, the more time required to train the transformer models. The online time is proportional to the number of entities: the more entities, the more time required to synthesize an ER dataset. The experiment results show that we can synthesize small ER datasets (*i.e.,*DBLP-ACM, Restaurant) within 5 hours and bigger ER datasets (*i.e.,*Walmart-Amazon, iTunes-Amazon) within 12 hours. The runtime is reasonable because it is totally fine to synthesize an ER dataset using hours which is much cheaper than preparing an ER dataset manually while keeping the same similarity distributions.

	DBLP-ACM	Restaurant	Walmart-Amazon	iTunes-Amazon
Offline	4.58 hour	3.45 hour	6.77 hour	9.83 hour
Online	4.02 min	1.57 min	36.85 min	78.94 min

TABLE IV
EFFICIENCY EVALUATION.

VIII. RELATED WORKS

Entity Resolution. Entity resolution (ER) is a classic problem in data management [1], [40], [41], [42], [43], [44]. Magellan [9] is an entity resolution system by traditional machine learning methods (*e.g.,* random forest, SVM, etc.). Deepmatcher [10] applies DL methods to ER inspired by the successes of DL in NLP task [23], [31]. ZeroER [12] proposes a generative model based on GMM for matching and non-matching distributions, and uses the EM algorithm to learn the distributions. Then ZeroER predicts whether an entity pair is matching by estimating their probability based on the distribution. The works [19], [45], [46] use crowdsourcing platforms to improve the accuracy of entity resolution.

ER Benchmarks. There is a widely known benchmark repository for entity resolution: The Magellan Data Repository [9]. It contains 13 collections of ER datasets, and each collection contains multiple ER benchmark datasets. Different collections of ER datasets are collected in different ways, or designed for different ER steps, data types, and domains.

Synthesized ER Benchmarks. EMBench [13], [14] is a benchmark for ER which aims to perform extensive evaluations of different ER algorithms. Given a collection of entities E, EMBench synthesizes new collections of entities E_1 by modifying the entities in E using predefined rules. EMBench can synthesize multiple new collections of entities (*i.e.,* E_1, E_2, E_3, etc.) by different combinations of rules, then users can test the accuracy and scalability of their ER algorithms on these synthesized benchmarks.

Generative Models. With the rise of DL, deep generative models (DGMs) [47], [38], [48], [29] have achieved tremendous success in images [49], [50], natural language processing [51], and speech recognition [52]. Recently, there have been several attempts [15], [16], [17], [18] of synthesizing relational data using GANs. These works only synthesize one relational table, and use GANs to learn the distribution of the original table. We want to synthesize two relational tables, and some entities in the two tables are similar (*i.e.,* matching). Therefore, the ER dataset synthesizing problem cannot be easily solved by GANs only.

IX. CONCLUSION

In this paper, we have proposed privacy preserving solutions to synthesize an ER dataset from a real ER dataset. We have conducted extensive experiments to verify that our approach can synthesize ER dataset such that the ER matcher trained on the synthesized dataset has similar performance with that of the real ER dataset.

ACKNOWLEDGMENT

This work is supported by NSF of China (61925205, 62102215, 62072261), Huawei, TAL education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation (2021M691784), Shuimu Tsinghua Scholar and Zhejiang Lab's International Talent Fund for Young Professionals.

REFERENCES

[1] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *TKDE*, vol. 19, no. 1, pp. 1–16, 2007.

[2] H. L. Dunn, "Record linkage," *American Journal of Public Health*, vol. 36, no. 12, pp. 1412–1416, 1946.

[3] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, "Automatic linkage of vital records," *Science*, vol. 130, no. 3381, pp. 954–959, 1959.

[4] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969.

[5] Y. Zhang, H. Ling, J. Gao, K. Yin, J.-F. Lafleche, A. Barriuso, A. Torralba, and S. Fidler, "Datasetgan: Efficient labeled data factory with minimal human effort," in *CVPR*, 2021, pp. 10 145–10 155.

[6] X. Yan, D. Acuna, and S. Fidler, "Neural data server: A large-scale search engine for transfer learning data," in *CVPR*, 2020, pp. 3893–3902.

[7] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler, "Meta-sim: Learning to generate synthetic datasets," in *CVPR*, 2019, pp. 4551–4560.

[8] S. Tripathi, S. Chandra, A. Agrawal, A. Tyagi, J. M. Rehg, and V. Chari, "Learning to generate synthetic data via compositing," in *CVPR*, 2019, pp. 461–470.

[9] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard *et al.*, "Magellan: Toward building entity matching management systems," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1197–1208, 2016.

[10] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *SIGMOD*. ACM, 2018, pp. 19–34.

[11] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proc. VLDB Endow.*, vol. 11, no. 11, pp. 1454–1467, 2018.

[12] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "Zeroer: Entity resolution using zero labeled examples," in *SIGMOD*, 2020, pp. 1149–1164.

[13] E. Ioannou, N. Rassadko, and Y. Velegrakis, "On generating benchmark data for entity matching," *J. Data Semant.*, vol. 2, no. 1, pp. 37–56, 2013.

[14] E. Ioannou and Y. Velegrakis, "Embench^{++}: Data for a thorough benchmarking of matching-related methods," *Semantic Web*, vol. 10, no. 2, pp. 435–450, 2019.

[15] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du, "Relational data synthesis using generative adversarial networks: A design space exploration," *VLDB*, vol. 13, no. 11, pp. 1962–1975, 2020.

[16] H. Chen, S. Jajodia, J. Liu, N. Park, V. Sokolov, and V. Subrahmanian, "Faketables: Using gans to generate functional dependency preserving tables with bounded real data." in *IJCAI*, 2019, pp. 2074–2080.

[17] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *VLDB*, vol. 11, no. 10, pp. 1071–1083, 2018.

[18] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional GAN," in *NeurIPS*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 7333–7343.

[19] C. Chai, G. Li, J. Li, D. Deng, and J. Feng, "Cost-effective crowdsourced entity resolution: A partial-order approach," in *SIGMOD*, 2016, pp. 969–984.

[20] J. Wang, G. Li, J. X. Yu, and J. Feng, "Entity matching: How similar is similar," *Proc. VLDB Endow.*, vol. 4, no. 10, pp. 622–633, 2011.

[21] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[22] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[24] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *SIGSAC*, 2016, pp. 308–318.

[25] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.

[26] F. Nicolas and E. Rivals, "Hardness results for the center and median string problems under the weighted and unweighted edit distances," *Journal of discrete algorithms*, vol. 3, no. 2-4, pp. 390–415, 2005.

[27] K. Aho, D. Derryberry, and T. Peterson, "Model selection for ecologists: the worldviews of aic and bic," *Ecology*, vol. 95, no. 3, pp. 631–636, 2014.

[28] A. P. Dempster, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. 39, 1977.

[29] I. Goodfellow, J. Pougetabadie, M. Mirza, B. Xu, D. Wardefarley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *NIPS*, pp. 2672–2680, 2014.

[30] L. Xu and K. Veeramachaneni, "Synthesizing tabular data using generative adversarial networks," *arXiv preprint arXiv:1811.11264*, 2018.

[31] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[32] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[35] Y. Govind, P. Konda, and et.al., "Entity matching meets data science: A progress report from the magellan project," in *SIGMOD*. ACM, 2019, pp. 389–403.

[36] P.-H. Lu, P.-C. Wang, and C.-M. Yu, "Empirical evaluation on synthetic data generation with generative adversarial network," in *WIMS*, 2019, pp. 1–6.

[37] J. M. Mateo-Sanz, F. Sebé, and J. Domingo-Ferrer, "Outlier protection in continuous microdata masking," in *International Workshop on Privacy in Statistical Databases*. Springer, 2004, pp. 201–215.

[38] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," *CoRR*, vol. abs/1802.06739, 2018.

[39] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Privbayes: Private data release via bayesian networks," *ACM Trans. Database Syst.*, vol. 42, no. 4, pp. 25:1–25:41, 2017.

[40] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: a survey," *The VLDB Journal*, vol. 29, no. 1, pp. 93–117, 2020.

[41] X. Qin, C. Chai, Y. Luo, T. Zhao, N. Tang, G. Li, J. Feng, X. Yu, and M. Ouzzani, "Interactively discovering and ranking desired tuples by data exploration," *The VLDB Journal*, pp. 1–25, 2022.

[42] X. Qin, C. Chai, Y. Luo, N. Tang, and G. Li, "Interactively discovering and ranking desired tuples without writing sql queries," in *SIGMOD*, 2020, pp. 2745–2748.

[43] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li, "Steerable self-driving data visualization," *TKDE*, vol. 34, no. 1, pp. 475–490, 2020.

[44] X. Qin, Y. Luo, N. Tang, and G. Li, "Deepeye: An automatic big data visualization framework," *Big data mining and analytics*, vol. 1, no. 1, pp. 75–82, 2018.

[45] C. Chai, G. Li, J. Li, D. Deng, and J. Feng, "A partial-order-based framework for cost-effective crowdsourced entity resolution," *VLDB J.*, vol. 27, no. 6, pp. 745–770, 2018.

[46] G. Li and C. C. et al., "CDB: optimizing queries with crowd-based selections and joins," in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 2017, pp. 1463–1478. [Online]. Available: https://doi.org/10.1145/3035918.3064036

[47] A. Oussidi and A. Elhassouny, "Deep generative models: Survey," in *ISCV*, 2018, pp. 1–8.

[48] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[49] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *CVPR*, 2019, pp. 4401–4410.

[50] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[51] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *AAAI*, 2017.

[52] C. Donahue, B. Li, and R. Prabhavalkar, "Exploring speech enhancement with generative adversarial networks for robust speech recognition," in *ICASSP*, 2018, pp. 5024–5028.