Deduplicated Sampling On-Demand

Luca Zecchini
BIFOLD & TU Berlin ★
Berlin, Germany
luca.zecchini@tu-berlin.de

Vasilis Efthymiou
Harokopio University
Athens, Greece
vefthym@hua.gr

Felix Naumann
Hasso Plattner Institute
Potsdam, Germany
felix.naumann@hpi.de

Giovanni Simonini
University of Modena
and Reggio Emilia, Italy
simonini@unimore.it

ABSTRACT

Data practitioners often sample their datasets to produce representative subsets for their downstream tasks. When entities in a dataset can be partitioned into multiple groups, stratified sampling is commonly used to produce subsets that match a target distribution, e.g., to select a balanced dataset for training a machine learning model. However, real-world data frequently contains duplicates — multiple representations of the same real-world entity — that can bias sampling, necessitating deduplication.

We define *deduplicated sampling* as the task of producing a clean sample from a dirty dataset according to a target distribution. The naïve approach to deduplicated sampling would first deduplicate the entire dataset upfront, then perform sampling *ex post*. However, that approach might be prohibitively expensive for large datasets and time/resource constraints. *Deduplicated sampling on-demand* with RadlER is a novel approach to produce a clean sample by focusing the cleaning effort only on entities required to appear in that sample. Our experimental evaluation, performed on multiple datasets from different domains, demonstrates that RadlER consistently outperforms the baseline approaches, providing data scientists with an efficient solution to quickly produce clean samples out of dirty data according to a target group distribution.

PVLDB Reference Format:
Luca Zecchini, Vasilis Efthymiou, Felix Naumann, and Giovanni Simonini. Deduplicated Sampling On-Demand. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dbmodena/radler.

1 SAMPLING DATA WITH DUPLICATES

Data practitioners often need to select representative data subsets to support their tasks, for instance to focus on exemplary instances when performing data visualization and exploration on large datasets [36, 50], or to prepare a balanced subset that adheres to a desired distribution for training machine learning models [8, 32] (e.g., balanced class labels for a classifier). This subset selection task is commonly known as *sampling* [75], a statistical process that aims to produce a representative subset out of a given population.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

★ Work partially done while at the University of Modena and Reggio Emilia, Italy.

_id	organization	address	year	source	amount
r_1	Big Apple Seeds	527 East 12th Street	2012	youth	4000.00
r_2	Man Up!	821 Van Siclen Avenue	2011	local	69000.00
r_3	Per Scholas, Inc.	804 East 138th Street	2013	local	30000.00
r_4	Man Up Inc.	821B Van Scilen Avenue	2012	local	50000.00
r_5	Man Up!, Inc.	821b Van Siclen Av.	2012	local	20000.00
r_6	St. Francis College	180 Remsen Street	2011	aging	9000.00

Figure 1: Excerpt inspired by the NYC funding dataset.

When a population can be considered as the combination of multiple distinct subpopulations (i.e., *groups*), defined on the basis of one or more features presented by its individuals (e.g., gender and/or ethnic group for people, brand for commercial products, etc.), *stratified sampling*, which performs sampling independently for each group, is widely used to guarantee that the produced sample follows a *target distribution* of individuals from the different groups. Typical target distributions can be defined based on the proportions of the groups in a reference population, as in the case of *demographic parity* [14], or even based on some fixed proportions [82], as in the case of *equal representation* [24], where each group is represented in the sample by the same number of individuals.

Being able to control the distribution of the various groups plays a key role in many real-world use cases. For instance, a proper representation of the different groups in the data is fundamental to prevent the insurgence of *bias* [49, 65] in analytics or in machine learning models trained on it, which might discriminate sensitive groups [59]. Balance or representativeness might also be enforced by external factors, such as laws or regulations [18, 45, 76].

Example 1.1. *Ellen, a data scientist, is doing a follow-up study on how organizations allocate discretionary New York City Council funding across different types of initiatives. She has a time constraint to prepare a report and the funding information can be found as open data in the NYC funding dataset, where initiatives are categorized by a source attribute (an example excerpt is shown in Figure 1). The analysis of every initiative requires acquiring financial reports from organizations and significant work to prepare the data, hence she cannot use the entire dataset and opts for limiting the analysis to 100 initiatives from distinct organizations selected with stratified sampling on the source (e.g., 15% youth, 5% aging, 20% local, etc.).*

Unfortunately, real data often presents quality issues [7], one of the most common being the presence of *duplicates*, i.e., multiple representations of the same real-world entity [54]. Duplicates often emerge when a dataset is generated by combining data acquired from multiple sources. Further, the different representations of an entity are often inconsistent, presenting missing or conflicting values for their features. The presence of duplicates, especially if inconsistent, can impact the outcome of downstream tasks significantly. For instance, it can alter the correctness of data analytics, potentially leading to poor data-driven business decisions that may cause additional undesired costs [33].

In the case of sampling, the negative effect of duplicates is twofold. Firstly, the produced sample inherits quality issues present in the original dataset. Secondly, duplicates distort the sampling process itself by introducing a bias in favor of entities described by multiple records, whose probability of appearing in the sample becomes consistently higher [26]. The presence of duplicates in the sample reduces its diversity and at the same time it can even amplify quality issues present in the original data, as the bias favors entities described by potentially inconsistent records.

Example 1.2. *The NYC funding dataset, introduced in Example 1.1 and described in detail in Section 4, contains duplicates: most organizations are represented by multiple records (e.g., records r_2, r_4, and r_5 in Figure 1). For instance, some organizations are described by up to 18 records each, while others have no duplicates at all. If Ellen performs stratified sampling over the dirty dataset, entities with more duplicates will likely be over-represented, as they appear up to 18 times more frequently than others, i.e., their chance to be picked is higher thanks to their duplicates.*

The detection of duplicates and their subsequent reconciliation represent the goal of *deduplication* [12]. Also known under the names of *record linkage*, *entity resolution*, and *duplicate detection*, this task is recognized as one of the longstanding challenges in data integration [21] and cleaning [35]. Given a *dirty dataset* — a dataset containing duplicates — as input, deduplication aims to produce the corresponding *clean dataset*, where each entity is described by a single *consolidated record* [19], obtained through the fusion of all duplicates describing that entity in the dirty dataset.

To prevent the issues caused by the presence of duplicates, practitioners are therefore required to produce a *clean sample*, i.e., a sample composed of consolidated records. We define the task of producing a clean sample from a dirty dataset according to a target distribution as *deduplicated sampling*. The naïve approach to deduplicated sampling — we denote it as *batch* [71] — requires to run deduplication on the entire dirty dataset upfront, then to perform sampling on the obtained clean dataset.

Yet, performing accurate detection of duplicates is often not trivial. State-of-the-art solutions rely on complex deep learning models [6] — including large language models [61] — to compare many records, and they may require the additional contribution of a domain expert to validate the results [41]. Thus, deduplication can be an expensive process in terms of time, computational resources, and therefore money, e.g., for using cloud resources or performing calls through the API of some large language model. Often, practitioners operate under time constraints and/or with limited resources [71], which makes the batch approach prohibitive, especially when dealing with large datasets.

For instance, in real-time fraud detection, swift analysis of user activity logs is essential. Cleaning entire logs increases response time, while duplicates may trigger false positives or mask anomalies. Thus, quickly extracting a clean sample allows timely and reliable detection. Similarly, in financial markets, immediate insights, news, and reports are crucial. Duplicates can obscure key trends, but a fast clean sampling enables prompt and accurate decisions. Also, in emergency response systems, rapidly extracting a clean sample

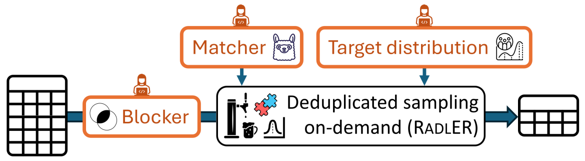


Figure 2: Deduplicated sampling on-demand with RadlER.

from noisy sensor data can be helpful for accurate real-time situational awareness. Further, even when affordable, cleaning the entire dataset to only use a (small) subset is extremely inefficient. A different approach, which limits the cleaning effort to the entities actually appearing in the sample, would allow to prevent additional costs and reduce the environmental impact of the exploited technologies [73].

Example 1.3. *Ellen cannot just sample 100 initiatives from the dirty dataset, as she would introduce bias (see Example 1.2). Thus, she needs to run deduplication. First, she exploits a blocking framework to detect candidate pairs of duplicate organizations. From the 16.3k records of the dataset, she ends up with 500k candidates to check — without blocking, she would need to check more than 144M candidates. Then, she exploits a popular large language model API to process the candidates: she instructs the prompt with some external knowledge about the organizations involved and asks to detect whether a candidate is an actual duplicate or not. Yet, with the pay-per-token model, processing each candidate costs approximately \$0.001, which leads to an overall cost of \$500 for deduplicating the entire dataset — even though Ellen just wants to sample 100 initiatives out of it.*

Contributions. For the first time in literature, we define the problem of deduplicated sampling, and we propose a novel *on-demand* approach to it, which enables practitioners to produce clean samples without cleaning the entire dirty data upfront. We implement *deduplicated sampling on-demand* through RadlER[1] (Figure 2), a dedicated solution that produces the clean sample incrementally by focusing the cleaning process on a single entity at a time.

For each of the specified groups, RadlER tracks all records that might describe an entity belonging to that group. Each record is assigned an updatable *weight*, computed as the tradeoff between the *cost* of obtaining the corresponding consolidated record — proportional to the number of comparisons with other records required to detect all of its duplicates — and its *benefit*, i.e., the probability that the described entity actually belongs to that group, based on the features of the record itself and its potential duplicates.

As the cleaning process focuses on a single entity at a time, RadlER performs deduplication in an iterative fashion. At the beginning of each iteration, it detects the group to which the next entity should belong to satisfy the target distribution required for the clean sample. Then, it picks a record among the ones tracked for that group through a weighted random selection and performs deduplication to produce the corresponding consolidated record. The use of weights makes the generation of the clean sample significantly cheaper, favoring the selection of records that require a small number of comparisons — hence correcting the bias introduced by

[1]Radler is a beverage obtained as a mix of beer and lemonade in variable proportions. Similarly, RadlER (with ER standing for entity resolution) produces samples by mixing different groups of entities according to the distribution required by the user.

the presence of duplicates — and whose entity is more likely to actually belong to the desired group.

Example 1.4. *Ellen has designed the blocking strategy, which identifies 500k candidate pairs of duplicate organizations, and the matching function, based on a large language model (see Example 1.3). Now, instead of deduplicating the entire dataset, she employs RadlER to sample 100 clean entities on-demand, as depicted in Figure 2. RadlER requires only 5k comparisons to produce the clean sample. Thus, the total cost is just about \$5 with RadlER, two orders of magnitude less than the clean-then-sample (batch) strategy from Example 1.3.*

We evaluate RadlER on multiple datasets with heterogeneous features covering different domains. The results of our experiments confirm that RadlER requires far fewer comparisons to produce clean samples from a dirty dataset than the batch approach (which requires deduplicating the entire data upfront), saving a significant amount of time and resources.

Outline. The remaining sections of the paper are structured as follows. In Section 2, we recall some key notions about deduplication and introduce the problem of deduplicated sampling, from which we consequently move to define deduplicated sampling on-demand. Then, we extensively describe RadlER in Section 3 and we report the results of its experimental evaluation in Section 4. After discussing related literature in Section 5, we present future directions for our research and conclude the paper with Section 6.

2 DEDUPLICATED SAMPLING ON-DEMAND

In this section, we first recall some key notions about deduplication in Section 2.1. Then, we provide the reader with a formal definition of deduplicated sampling in Section 2.2, from which we move to define deduplicated sampling on-demand in Section 2.3.

2.1 Deduplication

Deduplication [54] considers as input a *dirty dataset* $\mathcal{D}$ with schema $\mathcal{A}_{\mathcal{D}} = \{A_1, ..., A_m\}$ composed of $|\mathcal{D}|$ records. Each record $r \in \mathcal{D}$ can be represented as a tuple $r = (id, r[A_1], ..., r[A_m])$, where id is a unique record identifier and $r[A_i]$ is the (potentially null) value that record r assumes for attribute A_i. Deduplication aims to detect the disjoint clusters of *matching records* (or simply *matches*) in $\mathcal{D}$, each describing a distinct real-world entity. Then, it produces from every cluster of matches $\mathcal{M}_\varepsilon$ a single representative record for the described entity $\varepsilon = (id, \varepsilon[A_1], ..., \varepsilon[A_m])$, denoted as *consolidated record* [19] or, through a metonymy, directly as (cleaned) entity. We say therefore that each record $r \in \mathcal{M}_\varepsilon$ *describes* the entity ε or *refers* to ε. The produced set of consolidated records $\mathcal{D}^c$ is a *clean* version of the dirty dataset $\mathcal{D}$ and constitutes the output of deduplication. In this paper, we consider the traditional deduplication process for big data [13], which is composed of three major stages: blocking, entity matching, and data fusion.

To detect all duplicates, in principle every pair of records from $\mathcal{D}$ must be compared to determine whether they refer to the same real-world entity (i.e., whether they *match*). As comparing all pairs of records is often infeasible due to its inherently quadratic complexity, *blocking* [57] is used to make deduplication scalable. It determines a set $\mathcal{B}$ of *blocks*, i.e., possibly overlapping sets of records, and it is performed through a cheap *blocking function* (or simply *blocker*) $\beta : \mathcal{D} \rightarrow \mathbb{P}(\mathcal{B})$, based on some similarity criterion (e.g., sharing at least one token). Comparisons are then limited to those pairs of records that appear together in at least one block $b \in \mathcal{B}$. These pairs are denoted as *candidate matches* (or simply *candidates*) and compose a *candidate set* C of size $|C| \ll |\mathcal{D} \times \mathcal{D}|$. Post-processing techniques can be used to further reduce the number of candidates included in C, acting at the block level, such as block filtering and purging, or at the comparison level, such as meta-blocking [56]. By discarding obvious non-matches, blocking significantly reduces the number of comparisons while maintaining a high recall. Approaches presented in the literature range from naïve rule-based and overlap-based techniques [37] to state-of-the-art solutions built on TF/IDF [60], deep learning [10, 74], and meta-blocking [23, 28].

The *entity matching* stage [12] operates through a binary *matching function* (or simply *matcher*) $\mu : \mathcal{D} \times \mathcal{D} \rightarrow \{True, False\}$. For a pair of records in C, the matcher determines whether they *match* or not. Thus, entity matching detects the clusters of matching records in $\mathcal{D}$. Many different approaches to entity matching have been presented in the literature, such as rule-based matchers [72], solutions based on active learning [48] relying for instance on a human acting as an oracle [27] or on crowdsourcing [16, 31], and machine learning [37] or deep learning binary classifiers [22, 51], including state-of-the-art methods based on pre-trained language models such as BERT [43, 55] or large language models [52, 61, 63].

Finally, the generation of the consolidated record for the described entity ε from a cluster of matches $\mathcal{M}_\varepsilon$ is achieved through *data fusion* [9]. Data fusion relies on a *conflict resolution function* Φ, which associates to each attribute $A_i \in \mathcal{A}_{\mathcal{D}}$ an *aggregation function* ϕ_{A_i} that takes as input a list of values $[r[A_i], \forall\, r \in \mathcal{M}_\varepsilon]$ and returns a single value $\varepsilon[A_i]$. In other words, the aggregation function ϕ_{A_i} specifies how to resolve the possible inconsistencies of matching records for the attribute A_i. For instance, popular strategies involve majority voting (i.e., selecting the most frequent value), the choice of the maximum/minimum/average value, or a selection based on the record provenance, e.g., preferring values acquired from sources considered to be more reliable or updated.

2.2 Deduplicated Sampling

Sampling [75] is a statistical process that aims to produce a representative subset of a given population. A basic example of a sampling technique is *simple random sampling*, which produces a sample of size *n* from a population by randomly picking *n* individuals out of it. In many cases, a population can be considered as the combination of multiple distinct subpopulations (i.e., *groups*), defined on the basis of one or more features presented by its individuals. In this scenario, *stratified sampling* is widely used to guarantee that all groups are properly represented in the produced sample according to a *target distribution* (e.g., demographic parity). This technique first requires to partition the individuals into multiple disjoint groups (i.e., *strata*) in a process known as *stratification*, then sampling is performed independently for each stratum.

In the presence of duplicates, stratified sampling needs therefore to consider: *(i)* a set of disjoint groups Γ, which partitions the entities described by the dirty dataset $\mathcal{D}$ based on a set of categorical *sampling attributes* $\mathcal{A}_\Gamma \subseteq \mathcal{A}_{\mathcal{D}}$ (e.g., Γ = {{gender: "female", status: "single"}, ..., {gender: "male", status: "married"}} for $\mathcal{A}_\Gamma$ =

{gender, status}); *(ii)* a target distribution $d = [p(\gamma), \forall\, \gamma \in \Gamma]$, with $\sum_{p(\gamma)\in d} p(\gamma) = 1$, which defines as a probability $p(\gamma)$ the number of entities from each group $\gamma \in \Gamma$ required to appear in the produced sample $\mathcal{S}$. If no group is specified (i.e., $\Gamma = \emptyset$), a single *wildcard group* comprising all possible entities is considered, hence performing simple random sampling on the entire dataset.

We can now introduce the concepts of *clean sample* and *undistorted sample*, then build on them to define *deduplicated sampling*.

Definition 2.1 (Clean Sample). Given a dirty dataset $\mathcal{D}$, a sample $\mathcal{S}$ generated by deduplicating a set of records $\mathcal{D}_{\mathcal{S}} \subseteq \mathcal{D}$ is a *clean sample* of $\mathcal{D}$ if it is only composed of consolidated records. Namely, *(i)* every record $r \in \mathcal{S}$ describes a unique entity, i.e., $\mu(r, r') \rightarrow \mathit{False}, \forall\, r, r' \in \mathcal{S} \mid r \neq r'$, and *(ii)* every record $r \in \mathcal{S}$ is obtained through the fusion of all records describing that entity in $\mathcal{D}$, i.e., $\mu(r, \rho) \rightarrow \mathit{False}, \forall\, r \in \mathcal{S}, \forall\, \rho \in \mathcal{D} \setminus \mathcal{D}_{\mathcal{S}}$.

Definition 2.2 (Undistorted Sample). Given a target distribution d defined for a set of disjoint groups Γ, a sample $\mathcal{S}$ is *undistorted* with respect to d if the distribution $d_{\mathcal{S}}$ of its records over Γ has the minimum divergence from d among all samples of size $|\mathcal{S}|$, where $\mathit{divergence}(d, d') = \sum_{i=1}^{|\Gamma|} |d_i - d'_i|$.

Definition 2.3 (Deduplicated Sampling). Given a dirty dataset $\mathcal{D}$ and a target distribution d defined for a set of disjoint groups Γ, *deduplicated sampling* is the task of deduplicating $\mathcal{D}$ to produce a clean sample $\mathcal{S}$ of $\mathcal{D}$ that is undistorted with respect to d.

Deduplicated sampling is *agnostic* towards all deduplication functions defined in Section 2.1, i.e., the blocking function β, the matching function μ, and the conflict resolution function Φ, as it can operate with any function selected or defined by the user [71].

The naïve solution to deduplicated sampling, which we call *batch*, first performs deduplication on the entire dirty dataset $\mathcal{D}$ to obtain its clean version $\mathcal{D}^c$, then samples the consolidated records in $\mathcal{D}^c$ according to the target distribution d to produce an undistorted clean sample $\mathcal{S}$. The number of performed comparisons represents the *cost* κ of deduplicated sampling.

2.3 Deduplicated Sampling On-Demand

We can now define *deduplicated sampling on-demand*, our novel approach to run deduplicated sampling without cleaning the entire dirty data upfront, focusing instead the cleaning effort on the entities appearing in the clean sample $\mathcal{S}$, while retaining all properties of the sample as if it were taken from the clean dataset.

Definition 2.4 (Deduplicated Sampling On-Demand). Given a dirty dataset $\mathcal{D}$ and a target distribution d defined for a set of disjoint groups Γ, *deduplicated sampling on-demand* performs deduplicated sampling on $\mathcal{D}$ so to: *(i)* produce a clean sample $\mathcal{S}$ of $\mathcal{D}$ undistorted with respect to d, incrementally; *(ii)* cost $\kappa_{on\text{-}demand} \ll \kappa_{batch}$.

Deduplicated sampling on-demand focuses the cleaning effort on a single entity at a time to build the clean sample $\mathcal{S}$ incrementally. Thus, it performs deduplication according to the *on-demand* paradigm [71]. Note the difference to traditional *progressive* deduplication algorithms [58, 70, 79], which aim instead at prioritizing the evaluation of candidates that are most likely to match, regardless of the entities to which the records refer.

As the produced clean sample $\mathcal{S}$ is required to be undistorted at any time t of the process, deduplicated sampling on-demand inherently supports *early stopping* and *stop-and-resume* execution. If not stopped early, the process would generate the largest possible clean sample $\mathcal{S}$ undistorted with respect to the target distribution d, terminating when no more entities can be cleaned while maintaining the sample distribution $d_{\mathcal{S}}$ undistorted. However, deduplicated sampling on-demand can be stopped arbitrarily at any moment by the user, who can even define a stopping criterion expressed *a priori* with respect to a *budget* θ. In particular, depending on time constraints or available resources, users might require the process to stop as soon as it reaches the maximum cost κ_{max} (i.e., the maximum number of performed comparisons) that they are willing to spend to produce the clean sample $\mathcal{S}$. Further, additional early stopping techniques may require to terminate the process as soon as a defined number of cleaned entities $|\mathcal{S}|_{max}$ is inserted into $\mathcal{S}$ or its running time exceeds a timeout t_{max}.

For each matching function employed, it is possible to store (e.g., in a database) the detected matching and non-matching records for every record $r \in \mathcal{D}$. This avoids re-comparing candidates if multiple sampling operations are executed on $\mathcal{D}$. Alternatively, it is possible to directly store the produced consolidated records, which can replace their clusters of matches in $\mathcal{D}$ when using the same deduplication functions, hence cleaning $\mathcal{D}$ incrementally.

3 RADLER

In this section, we describe how RadlER performs deduplicated sampling on-demand. In particular, we provide a complete overview of the algorithm in Section 3.1, while Section 3.2 presents the adopted weighting scheme. Then, Sections 3.3–3.5 delve into the details of three specific steps of the presented algorithm.

3.1 Algorithm Overview

To produce an undistorted clean sample $\mathcal{S}$, RadlER (Algorithm 1) receives as input: *(i)* a dirty dataset $\mathcal{D}$; *(ii)* a hash table $\mathcal{N}$, associating each record to the set of its *neighbors*, i.e., its candidate matches previously obtained through a blocking function β (towards which the algorithm is agnostic); *(iii)* a set of disjoint groups Γ, partitioning entities based on the values of the sampling attributes $\mathcal{A}_{\Gamma}$; *(iv)* a target distribution d for the groups in Γ; *(v)* optionally, a budget θ and/or a maximum sample size $|\mathcal{S}|_{max}$, to perform early stopping as soon as θ comparisons have been performed or $|\mathcal{S}|_{max}$ cleaned entities have been inserted into $\mathcal{S}$, respectively. Note that the hash table $\mathcal{N}$ can be made available in a key-value store, such as Redis or RocksDB, to efficiently address memory constraints. Figure 3 depicts RadlER operating on an example dataset about people. An excerpt from the dataset $\mathcal{D}$ and the hash table $\mathcal{N}$ is shown in Figure 3a. Here, gender and status are considered as sample attributes, while equal representation is the target distribution.

3.1.1 Setup. At the beginning of the algorithm, the clean sample $\mathcal{S}$ is initialized as an empty set and the consumed budget θ' (i.e., the number of performed comparisons) is set to 0 (Line 1). In accordance with Definition 2.4, RadlER focuses the cleaning effort on a single entity at a time to populate $\mathcal{S}$ incrementally. Thus, deduplicated sampling on-demand can be seen as an iterative process, which cleans at each iteration τ a randomly selected entity ε_τ belonging

Algorithm 1: RadlER algorithm

```
Input: Dataset $\mathcal{D}$, neighbors $\mathcal{N}$, groups $\Gamma$, target distribution $d$,
       budget $\theta$ (default $\infty$), max sample size $|\mathcal{S}|_{max}$ (default $\infty$)
Output: Undistorted clean sample $\mathcal{S}$
1  $\mathcal{S} \leftarrow \emptyset,\ \theta' \leftarrow 0$          // clean sample and consumed budget
2  $\mathcal{R}, \mathcal{G} \leftarrow$ setup$(\mathcal{D}, \mathcal{N}, \Gamma)$   // record sketches and group records
3  while $|\mathcal{S}| < |\mathcal{S}|_{max}$ do
4      $\hat{\gamma}_\tau \leftarrow$ selectTargetGroup$(\mathcal{S}, \mathcal{G}, d, \Gamma)$        // target group
5      if $\hat{\gamma}_\tau = None$ then
6          break                                     // prevent distortion
7      $p_\tau.id \leftarrow$ weightedRandom$(\mathcal{G}[\hat{\gamma}_\tau])$     // pivot record ID
8      $s_{p_\tau} \leftarrow \mathcal{R}[p_\tau.id]$                     // pivot record sketch
       // Clean the entity described by the pivot record
9      $\mathcal{K} \leftarrow \emptyset,\ \tilde{\mathcal{M}} \leftarrow \emptyset$              // comparisons and non-matches
10     if not $s_{p_\tau}.clean$ then
11         $s_{p_\tau}.\mathcal{M}, \mathcal{K}, \theta' \leftarrow$ match$(p_\tau.id, s_{p_\tau}.\mathcal{M}, \mathcal{K}, \mathcal{D}, \mathcal{N}, \theta, \theta')$
12         if $\theta' > \theta$ then
13             break                                 // budget exceeded
14         $\tilde{\mathcal{M}} \leftarrow$ getRecordIds$(\mathcal{K}) \setminus s_{p_\tau}.\mathcal{M}$
15         $s_{p_\tau}.clean \leftarrow True$                  // entity matching done
16     $\mathcal{R}, \mathcal{G}, s_{p_\tau} \leftarrow$ update$(\mathcal{R}, \mathcal{G}, s_{p_\tau}, \tilde{\mathcal{M}}, \mathcal{D}, \mathcal{N}, \hat{\gamma}_\tau, \Gamma)$
17     if $s_{p_\tau}.\gamma = \hat{\gamma}_\tau$ then
18         $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_{p_\tau}.\varepsilon\}$        // insert into the clean sample
19 return $\mathcal{S}$
```

to a group $\gamma \in \Gamma$ that allows to maintain the sample undistorted with respect to the target distribution d. This group is therefore denoted as the *target group* $\hat{\gamma}_\tau$ for iteration τ.

The selection of the entity ε_τ to clean at iteration τ has to be performed on the original records in $\mathcal{D}$. Thus, to focus the cleaning effort on an entity that actually belongs to $\hat{\gamma}_\tau$, we need to know for each record $r \in \mathcal{D}$ to which groups the described entity ε_r *might* belong. Further, we would like to favor the selection of cheaper entities (i.e., requiring fewer comparisons) over more expensive ones, to mitigate the bias caused by duplicates, which would favor the selection of entities represented by more records.

To this end, we maintain a hash table $\mathcal{G}$ to track for each group $\gamma \in \Gamma$ all records that might describe an entity belonging to γ, whose identifiers are stored in a further hash table $\mathcal{G}[\gamma]$. Within $\mathcal{G}[\gamma]$, each record r is associated to an updatable *weight* ω_γ^r, used to perform the *weighted* random selection of the entity to clean. The weight ω_γ^r represents the tradeoff between the estimated probability that ε_r belongs to γ (i.e., the *benefit* of cleaning ε_r) and the estimated number of comparisons required for cleaning it (i.e., its *cost*). The weight ω_γ^r can assume a value in $[0, 1]$. If $\omega_\gamma^r = 0$, it is impossible that ε_r belongs to γ, hence r is not represented in $\mathcal{G}[\gamma]$. If $\omega_\gamma^r = 1$, then ε_r certainly belongs to γ and it does not require further comparisons to detect the matches, hence r is associated to the maximum weight as the *cheapest* possible selection. We describe the weighting scheme in detail in Section 3.2. In the example, record r_2 has r_1 and r_3 as neighbors, as shown in Figure 3a. As two records belong to group 0 (married females) and one to group 1 (single females), the entity ε_{r_2} might belong to either of those groups, hence r_2 appears in both $\mathcal{G}[0]$ (with greater weight) and $\mathcal{G}[1]$ in Figure 3b. Record r_4, which has no neighbors, is inserted only into $\mathcal{G}[0]$ with weight 1.

In addition to $\mathcal{G}$, a hash table $\mathcal{R}$ is used to associate to each record $r \in \mathcal{D}$ a *sketch* s_r, i.e., an object that stores relevant information about r, namely: *(i)* the represented cluster of matching records $\mathcal{M}_r$ (initialized with $r.id$ itself and updated when running entity matching)[2]; *(ii)* a flag (*clean*) to determine whether all matches of r have already been identified or entity matching is needed; *(iii)* a hash table Ω_r, which stores the weight ω_γ^r associated to r for each group $\gamma \in \Gamma$; *(iv)* the cleaned entity ε_r and its group γ_r, both initialized to *None* and updated when running data fusion. For instance, the sketch s_{r_2} for record r_2 is shown in Figure 3c. $\mathcal{R}$ and $\mathcal{G}$ are initialized at the beginning of the algorithm (Line 2) through the **setup**() function (Function 1), described in Section 3.3.

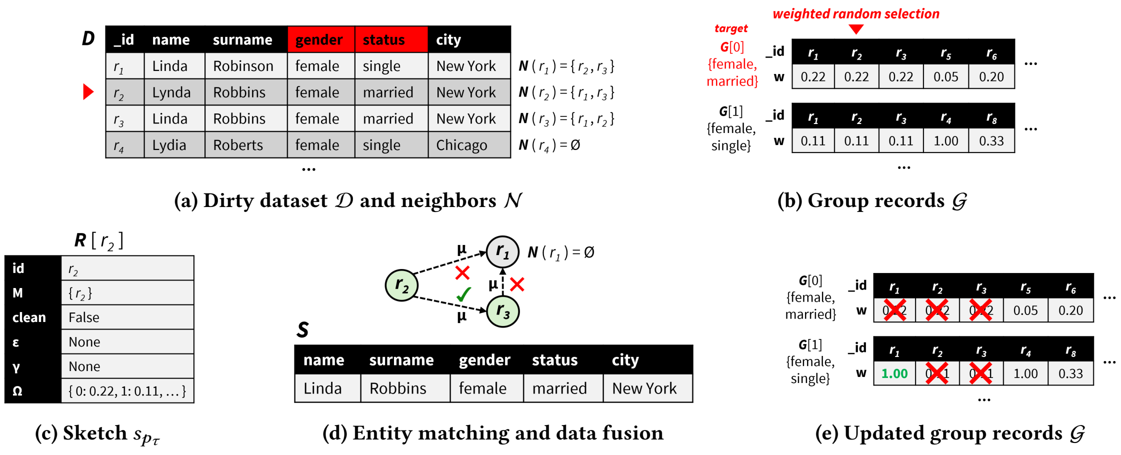
3.1.2 *Iteration.*

After initializing data structures, RadlER starts its iterations to populate the clean sample $\mathcal{S}$ incrementally, until the maximum sample size $|\mathcal{S}|_{max}$ is reached (Line 3). If no value is specified for the size (hence $|\mathcal{S}|_{max} = \infty$), RadlER produces the largest possible clean sample $\mathcal{S}$ that is undistorted with respect to the target distribution d. The selection of the target group $\hat{\gamma}_\tau$ for iteration τ (Line 4) is performed in two steps. First, we detect the groups that allow to maintain $\mathcal{S}$ undistorted with respect to d. If $d_{\tau-1}$ is the current distribution of the entities in $\mathcal{S}$ and d_τ^γ the one obtained by adding to $\mathcal{S}$ an entity from group γ, we select the subset $\tilde{\Gamma} = \{\tilde{\gamma} \in \Gamma \mid divergence(d_\tau^{\tilde{\gamma}}, d) \leq divergence(d_\tau^\gamma, d), \forall \gamma \in \Gamma\}$. Then, we filter out the groups for which $\mathcal{G}[\gamma]$ is empty (i.e., no more entities to clean). Thus, we obtain $\mathring{\Gamma} = \{\gamma \in \tilde{\Gamma} \mid |\mathcal{G}[\gamma]| > 0\}$. If $\mathring{\Gamma}$ contains multiple groups, different tie-breaking strategies can be used to pick $\hat{\gamma}_\tau$ out of $\mathring{\Gamma}$, e.g., select a random group or the one with the maximum average weight in $\mathcal{G}[\gamma]$. If $\mathring{\Gamma} = \emptyset$, it is not possible to add further entities while maintaining $\mathcal{S}$ undistorted, hence $\hat{\gamma}_\tau$ is set to *None* and the iterations terminate (Lines 5-6).

The entity to be cleaned is selected by picking a record through a weighted random selection out of $\mathcal{G}[\hat{\gamma}_\tau]$. For instance, in Figure 3b the target group $\hat{\gamma}_\tau$ is group 0, hence a weighted random selection is performed on $\mathcal{G}[0]$, picking r_2. As this record guides the cleaning process for iteration τ, it is denoted as the *pivot record* p_τ (Line 7). Its sketch s_{p_τ} is then retrieved from $\mathcal{R}$ (Line 8). Two cases are possible for p_τ: *(i)* it presents candidate matches in $\mathcal{N}$ (i.e., $|\mathcal{N}[p_\tau.id]| > 0$) to verify through entity matching; *(ii)* the cluster of matches to which it belongs has already been discovered completely (i.e., $|\mathcal{N}[p_\tau.id]| = 0$), hence entity matching is not needed. The sketch flag *clean*, set to *True* if entity matching is not needed, discriminates between the two cases (Line 10).

If the pivot record p_τ requires entity matching, its matches are detected among its neighbors through the **match**() function (Function 2), described in detail in Section 3.4, and their identifiers are added to $\mathcal{M}_{p_\tau}$ (Line 11), which was previously populated by $p_\tau.id$ only. In Figure 3d, r_3 is a match of r_2, hence r_3 is added to $\mathcal{M}_{p_\tau}$. The performed comparisons are stored into the set $\mathcal{K}$, previously initialized as empty (Line 9), to avoid presenting the same pair of records to the matcher multiple times. Further, $\mathcal{K}$ is then used to retrieve the identifiers of those records that were compared to p_τ but turned out to describe a different entity, i.e., the *non-matches* of p_τ. Non-matches are stored in a dedicated set $\tilde{\mathcal{M}}$ (Line 14), which

[2]Note that in the object attribute notation (as for instance in Algorithm 1) we consider the subscript implicit to avoid redundancy, e.g., $\mathcal{M}_r$ is equivalent to $s_r.\mathcal{M}$.



D

_id	name	surname	gender	status	city	
r_1	Linda	Robinson	female	single	New York	$N(r_1) = \{r_2, r_3\}$
r_2	Lynda	Robbins	female	married	New York	$N(r_2) = \{r_1, r_3\}$
r_3	Linda	Robbins	female	married	New York	$N(r_3) = \{r_1, r_2\}$
r_4	Lydia	Roberts	female	single	Chicago	$N(r_4) = \emptyset$
...						

(a) Dirty dataset $\mathcal{D}$ and neighbors $\mathcal{N}$

	_id	r_1	r_2	r_3	r_5	r_6	
G[0] {female, married}	w	0.22	0.22	0.22	0.05	0.20	...

	_id	r_1	r_2	r_3	r_4	r_8	
G[1] {female, single}	w	0.11	0.11	0.11	1.00	0.33	...

...

(b) Group records $\mathcal{G}$

R $[r_2]$

id	r_2
M	$\{r_2\}$
clean	False
ε	None
γ	None
Ω	$\{0: 0.22, 1: 0.11, \dots\}$

(c) Sketch s_{p_τ}

S

name	surname	gender	status	city
Linda	Robbins	female	married	New York

(d) Entity matching and data fusion

	_id	r_1	r_2	r_3	r_5	r_6	
G[0] {female, married}	w	$\cancel{0.22}$	$\cancel{0.22}$	$\cancel{0.22}$	0.05	0.20	...

	_id	r_1	r_2	r_3	r_4	r_8	
G[1] {female, single}	w	1.00	$\cancel{0.11}$	$\cancel{0.11}$	1.00	0.33	...

...

(e) Updated group records $\mathcal{G}$

Figure 3: RadlER in action on an example dataset about people, with gender and status as sampling attributes and equal representation as the target distribution.

was also previously initialized as empty (Line 9). If the number of performed comparisons exceeds the budget θ, entity matching is stopped immediately (see Function 2 for more details) and the iterations terminate (Lines 12-13). After performing entity matching, the sketch flag *clean* is set to *True* (Line 15).

At this point, the sketch s_{p_τ} and the hash tables $\mathcal{R}$ and $\mathcal{G}$ are updated (Line 16) through the dedicated **update**() function (Function 3). First, for each non-match in $\tilde{\mathcal{M}}$, its neighbors are updated by removing p_τ and its matches, hence its weights are recomputed and its representations in $\mathcal{R}$ and $\mathcal{G}$ are updated accordingly. The consolidated record for the entity ε_{p_τ} is produced from the matches in $\mathcal{M}_{p_\tau}$ through data fusion, and the group γ_{p_τ} to which it actually belongs is finally detected. If ε_{p_τ} does not belong to any of the groups in Γ, γ_{p_τ} is set to *None*. Then, ε_{p_τ} and γ_{p_τ} are stored into the dedicated attributes of the sketch s_{p_τ}. Further, the matches in $\mathcal{M}_{p_\tau}$ are removed from $\mathcal{R}$ and $\mathcal{G}$, as they should not be considered as independent records anymore. If γ_{p_τ} corresponds to the target group $\hat{\gamma}_\tau$, then ε_{p_τ} is inserted into $\mathcal{S}$ (Lines 17-18); otherwise, the updated sketch s_{p_τ} is reinserted into $\mathcal{R}$ and its identifier into $\mathcal{G}[\gamma_{p_\tau}]$ with a weight equal to 1, where it is available to be selected in one of the following iterations. In Figure 3, as ε_{p_τ} (produced from r_2 and r_3) belongs to $\hat{\gamma}_\tau$, it is inserted into $\mathcal{S}$ (Figure 3d). r_2 and r_3 are therefore removed from $\mathcal{G}$, while r_1 — whose set of neighbors is now empty, as shown in Figure 3d — is maintained only in $\mathcal{G}[1]$ with weight 1 (Figure 3e). All details about the update operations are provided in Section 3.5. Finally, when its generation is completed, RadlER returns the undistorted clean sample $\mathcal{S}$ (Line 19).

Considering the memory overhead introduced by data structures, hash table $\mathcal{R}$ is initially composed of $|\mathcal{D}|$ sketches, each dominated by the hash table Ω_r of size $|\Gamma|$, for a space complexity of $O(|\mathcal{D}| \cdot |\Gamma|)$. Updates to $\mathcal{M}_r$ and ε_r can increase the sketch size. However, consolidated records can be easily maintained in a database and accessed through an identifier, avoiding their direct storage in ε_r. Further, when $\mathcal{M}_r$ increases, some other sketches are set to *None*. Thus, the memory occupation of $\mathcal{R}$ decreases throughout the process. Regarding hash table $\mathcal{G}$, if every record initially appears in each of the $|\Gamma|$ inner hash tables, its space complexity is $O(|\mathcal{D}| \cdot |\Gamma|)$. Note that in our experiments (Section 4) a record appears on average in only about 1/3 of the inner hash tables.

3.2 Weighting Scheme

The weight ω_γ^r, associated to the record r for the group $\gamma \in \Gamma$, is computed by taking into account all records that might contribute to determine the values assumed by the entity ε_r, hence the records whose identifier appears in the set $\mathcal{N}'_r = \{r.id\} \cup \mathcal{N}[r.id]$, i.e., r itself and its potential duplicates.

The cost of cleaning ε_r is directly proportional to the size of $\mathcal{N}'_r$. As weights aim to favor records that determine fewer comparisons, mitigating the bias introduced by duplicates, the cost contributes to ω_γ^r with a factor $1/|\mathcal{N}'_r|$, whose value is equal to 1 when $\mathcal{N}'_r$ only contains the identifier of r (i.e., no comparisons needed). The benefit of cleaning ε_r is aimed to reflect instead the probability that ε_r actually belongs to γ, which is equal to 1 when all records in $\mathcal{N}'_r$ assume the values that determine membership in γ for the sampling attributes $\mathcal{A}_\Gamma$. In particular, we can evaluate this probability independently for each attribute $\alpha \in \mathcal{A}_\Gamma$. We denote as $\sigma_\alpha(\mathcal{N}'_r)$ the subset of records in $\mathcal{N}'_r$ that satisfy the condition required on the attribute α to belong to γ and we can estimate this probability for the attribute α as $|\sigma_\alpha(\mathcal{N}'_r)|/|\mathcal{N}'_r|$. Note that records for which α is null can be ignored when computing $|\mathcal{N}'_r|$. Considering the contribution of the cost and the probability for every attribute $\alpha \in \mathcal{A}_\Gamma$, the weight ω_γ^r can be therefore computed as follows:

$$\omega_\gamma^r = \frac{1}{|\mathcal{N}'_r|} \cdot \prod_{\alpha \in \mathcal{A}_\Gamma} \frac{|\sigma_\alpha(\mathcal{N}'_r)|}{|\mathcal{N}'_r|} \qquad (1)$$

For instance, record r_2 in Figure 3 has two neighbors: r_1 and r_3. Thus, its cost is $1/3 = 0.33$. As r_2 and r_3 belong to group 0 and r_1 to group 1 based on the status attribute, the benefit is $1 \cdot 2/3 = 0.67$ for group 0, for a weight of $0.33 \cdot 0.67 = 0.22$, and $1 \cdot 1/3 = 0.33$ for group 1, for a weight of $0.33 \cdot 0.33 = 0.11$.

Note that Equation 1 assumes all records in $\mathcal{N}'_r$ to be equally relevant to r. Nevertheless, in some cases it is possible to determine a *relevance score* λ_ρ with respect to r for each record $\rho \in \mathcal{N}'_r$, such as the matching probability returned by meta-blocking for

Function 1: setup() function

```
Input: Dataset $\mathcal{D}$, neighbors $\mathcal{N}$, groups $\Gamma$
Output: Record sketches $\mathcal{R}$, group records $\mathcal{G}$
1  $\mathcal{R} \leftarrow \emptyset,\ \mathcal{G} \leftarrow \emptyset$                // empty data structures
2  forall $r$ in $\mathcal{D}$ do
       // Generate the record sketch $s_r$
3      $s_r.id \leftarrow r.id,\ s_r.\mathcal{M} \leftarrow \{r.id\}$  // ID and cluster of matches
4      $s_r.clean \leftarrow$ len$(\mathcal{N}[r.id]) = 0$        // no further matches
5      $s_r.\varepsilon \leftarrow None,\ s_r.\gamma \leftarrow None$   // clean entity and its group
6      $s_r.\Omega \leftarrow$ computeWeights$(\mathcal{N}[r.id] \cup \{r.id\}, \mathcal{D}, \Gamma)$
7      $\mathcal{R}[r.id] \leftarrow s_r$                    // store record sketch
       // Insert the record ID into the group hash tables
8      forall $\gamma$ in $\Gamma$ do
9          if $s_r.\Omega[\gamma] > 0$ then
10             $\mathcal{G}[\gamma][r.id] \leftarrow s_r.\Omega[\gamma]$
11 return $\mathcal{R}, \mathcal{G}$
```

Function 2: match() function

```
Input: Record identifier $r.id$, matches $\mathcal{M}$, comparison tracker $\mathcal{K}$,
       dataset $\mathcal{D}$, neighbors $\mathcal{N}$, budget $\theta$, consumed budget $\theta'$
Output: Updated versions of $\mathcal{M}, \mathcal{K}, \theta'$
1  forall $n.id$ in $\mathcal{N}[r.id]$ do
2      if $n.id$ in $\mathcal{M}$ then
3          continue    // neighbor already identified as a match
4      if $(r.id, n.id)$ in $\mathcal{K}$ or $(n.id, r.id)$ in $\mathcal{K}$ then
5          continue              // comparison already performed
6      $\theta' \leftarrow \theta' + 1$               // increment consumed budget
7      if $\theta' > \theta$ then
8          break                              // budget exceeded
       // Perform the comparison
9      $\mathcal{K} \leftarrow \mathcal{K} \cup \{(r.id, n.id)\}$        // update comparison tracker
10     if $\mu(r.id, n.id, \mathcal{D})$ then
11         $\mathcal{M} \leftarrow \mathcal{M} \cup \{n.id\}$                  // update matches
12         $\mathcal{M}, \mathcal{K}, \theta' \leftarrow$ match$(n.id, \mathcal{M}, \mathcal{K}, \mathcal{D}, \mathcal{N}, \theta, \theta')$
13 return $\mathcal{M}, \mathcal{K}, \theta'$
```

the candidate (r, ρ). In this case, each record contributes to the probability depending on its relevance score, hence the numerator of the latter term is computed instead as $\sum_{\rho \in \sigma_\alpha(\mathcal{N}'_r)} \lambda_\rho$.

3.3 Initializing Data Structures

The **setup**() function (Function 1) illustrates the initialization and the population of the hash tables $\mathcal{R}$ and $\mathcal{G}$. The function receives as input the dirty dataset $\mathcal{D}$, the hash table $\mathcal{N}$, which associates each record to the set of its neighbors, and the set of disjoint groups Γ. It produces as output $\mathcal{R}$ and $\mathcal{G}$, both initialized as empty hash tables (Line 1). As pointed out in Section 3.1, $\mathcal{R}$ contains a sketch s_r generated for every record $r \in \mathcal{D}$, while $\mathcal{G}$ tracks for each group $\gamma \in \Gamma$ all records that might describe an entity that belongs to γ, whose identifiers are stored in a further hash table $\mathcal{G}[\gamma]$.

For every record $r \in \mathcal{D}$, Function 1 produces a sketch s_r, which stores the information about r used throughout Algorithm 1. First, s_r stores the identifier of r and the represented cluster of matches $\mathcal{M}_r$, initialized with $r.id$ itself (Line 3). If the set of its neighbors $\mathcal{N}[r.id]$ is empty, only r itself describes the entity ε_r. Thus, entity matching is not needed and the flag *clean* is set to *True* (Line 4). The attributes ε_r and γ_r, respectively used to store the consolidated record for the described entity and its actual group, are both initialized to *None* (Line 5). Finally, the weight ω^r_γ associated to r for each group $\gamma \in \Gamma$ is computed as described in Section 3.2 and stored into the hash table Ω_r (Line 6). Once created, the sketch s_r is stored into $\mathcal{R}$, where it can be accessed through its identifier (Line 7). Further, its identifier is inserted into the hash table $\mathcal{G}[\gamma]$ (associated to its weight $\Omega_r[\gamma]$) for every group $\gamma \in \Gamma$ to which the entity ε_r might belong, i.e., for which $\Omega_r[\gamma] > 0$ (Lines 8-10). Finally, the populated hash tables $\mathcal{R}$ and $\mathcal{G}$ are returned by Function 1 (Line 11).

3.4 Performing Entity Matching

The **match**() function (Function 2) shows how RadlER performs entity matching. Given a record r, the function aims to detect its matches by comparing r to its neighbors. Pairs of records are compared through the matcher μ, towards which RadlER is agnostic. The function receives as input the identifier of the consider record $r.id$, the sets $\mathcal{M}$ and $\mathcal{K}$, respectively tracking the detected matches and the performed comparisons, the dirty dataset $\mathcal{D}$ (used by the matcher to access the attribute values), the hash table $\mathcal{N}$ associating each record to its neighbors, the budget θ, and the consumed budget θ', incremented every time a comparison is performed and returned as output together with the updated sets $\mathcal{M}$ and $\mathcal{K}$.

Function 2 iterates on the identifiers of the neighbors (Line 1). For each neighbor n, we first check whether it is actually needed to compare it to r. A comparison can be skipped if: *(i)* n was already identified as a match of r (Lines 2-3); *(ii)* r and n were already compared and turned out not to match (Lines 4-5). Further, if performing the comparison would exceed the budget θ, the iterations terminate (Lines 6-8) and Algorithm 1 consequently stops the cleaning process, as described in Section 3.1.

When a comparison has to be performed, the pair of identifiers $(r.id, n.id)$ is added to $\mathcal{K}$ (Line 9) and the matcher μ is used to determine if r and n describe or not the same real-world entity (Line 10). If they are considered to match (i.e., the matcher returns *True* as the result of their comparison), the identifier of n is inserted into the set of matches $\mathcal{M}$ (Line 11), while the **match**() function is called recursively for n to detect further matches among its neighbors (Line 12). This is required to ensure the discovery of all matches of r, as — depending on the blocking function β — some neighbors of n might not appear among the ones of r. Finally, the updated sets $\mathcal{M}$ and $\mathcal{K}$ are returned together with the consumed budget θ', incremented to track all performed comparisons (Line 13).

3.5 Updating Data Structures

The **update**() function (Function 3) shows how the hash tables $\mathcal{R}$ and $\mathcal{G}$ and the pivot record sketch s_{p_τ} are updated at the end of each iteration of Algorithm 1. Beyond $\mathcal{R}$, $\mathcal{G}$, and s_{p_τ}, which are then returned as output in their updated versions, the function receives as input the set of non-matches $\tilde{\mathcal{M}}$, the dirty dataset $\mathcal{D}$, the hash table $\mathcal{N}$ associating each record to its neighbors, the target group $\hat{\gamma}_\tau$, and the set of disjoint groups Γ.

First, for every non-match $\tilde{m} \in \tilde{\mathcal{M}}$ (Line 1), its neighbors are updated by removing records appearing in $\mathcal{M}_{p_\tau}$, as they certainly

Function 3: update() function

```
Input: Sketches $\mathcal{R}$, group records $\mathcal{G}$, sketch $s_{p_\tau}$, non-matches $\tilde{\mathcal{M}}$,
       dataset $\mathcal{D}$, neighbors $\mathcal{N}$, target group $\hat{\gamma}_\tau$, groups $\Gamma$
Output: Updated versions of $\mathcal{R}$, $\mathcal{G}$, $s_{p_\tau}$
// Update neighbors and weights for non-matching records
1  forall $\tilde{m}.id$ in $\tilde{\mathcal{M}}$ do
2      $\mathcal{N}_{\tilde{m}} \leftarrow \mathcal{N}[\tilde{m}.id] \setminus s_{p_\tau}.\mathcal{M}$, $\mathcal{N}[\tilde{m}.id] \leftarrow \mathcal{N}_{\tilde{m}}$
3      $\mathcal{R}[\tilde{m}.id].\Omega \leftarrow$ computeWeights($\mathcal{N}_{\tilde{m}} \cup \{\tilde{m}.id\}, \mathcal{D}, \Gamma$)
4      $\mathcal{G} \leftarrow$ setWeights($\mathcal{G}, \tilde{m}.id, \mathcal{R}[\tilde{m}.id].\Omega$)
// Produce the clean entity and replace matching records
5  if $s_{p_\tau}.\varepsilon = None$ then
6      $s_{p_\tau}.\varepsilon \leftarrow \Phi(s_{p_\tau}.\mathcal{M}, \mathcal{D})$        // clean entity (data fusion)
7      $s_{p_\tau}.\gamma \leftarrow$ checkGroup($s_{p_\tau}.\varepsilon$)              // entity group
8      $s_{p_\tau}.\Omega \leftarrow$ updateWeights($s_{p_\tau}.\gamma, \Gamma$)              // weights
9      forall $m.id$ in $s_{p_\tau}.\mathcal{M}$ do
10         $\mathcal{N}[m.id] \leftarrow \emptyset$
11         $\mathcal{R}[m.id] \leftarrow None$, $\mathcal{G} \leftarrow$ dropWeights($\mathcal{G}, m.id$)
12     if $s_{p_\tau}.\gamma \neq \hat{\gamma}_\tau$ then
13         $\mathcal{R}[s_{p_\tau}.id] \leftarrow s_{p_\tau}$, $\mathcal{G} \leftarrow$ setWeights($\mathcal{G}, s_{p_\tau}.id, s_{p_\tau}.\Omega$)
14 else
15     if $s_{p_\tau}.\gamma = \hat{\gamma}_\tau$ then
16         $\mathcal{R}[s_{p_\tau}.id] \leftarrow None$, $\mathcal{G} \leftarrow$ dropWeights($\mathcal{G}, s_{p_\tau}.id$)
17 return $\mathcal{R}, \mathcal{G}, s_{p_\tau}$
```

do not describe the same entity $\varepsilon_{\tilde{m}}$ (Line 2). Weights are therefore recomputed and stored into the corresponding sketch (Line 3). Further, in every hash table $\mathcal{G}[\gamma]$ where $\tilde{m}$ is present, the associated weight is updated accordingly — if the new weight $\omega_\gamma^{\tilde{m}} = 0$, then $\tilde{m}$ is dropped from $\mathcal{G}[\gamma]$ (Line 4).

If the consolidated record for the entity described by the pivot record p_τ has not been generated yet[3] (Line 5), ε_{p_τ} is obtained from the matches in $\mathcal{M}_{p_\tau}$ through the defined conflict resolution function Φ, towards which RadlER is agnostic (Line 6). Thus, it is finally possible to detect the group γ_{p_τ} to which ε_{p_τ} actually belongs (Line 7), then set a weight in Ω_{p_τ} equal to 1 for the group γ_{p_τ} — if not *None* — and to 0 for all other groups in Γ (Line 8). The matches in $\mathcal{M}_{p_\tau}$ are considered from now on as a single entity, represented through the updated pivot record p_τ. Thus, their sets of neighbors are emptied and they are removed from $\mathcal{R}$ and $\mathcal{G}$ (Lines 9-11).

If the entity ε_{p_τ} does not belong to the target group $\hat{\gamma}_\tau$, the updated sketch s_{p_τ} is reinserted into $\mathcal{R}$ and its identifier is added to $\mathcal{G}[\gamma_{p_\tau}]$ — if γ_{p_τ} is not *None* — to allow its selection in one of the next iterations (Lines 12-13). Otherwise, it is added to the clean sample $\mathcal{S}$ by Algorithm 1 as soon as Function 3 terminates, and p_τ is removed from $\mathcal{R}$ and $\mathcal{G}$ if it is present there (Lines 15-16). Finally, the updated versions of the sketch s_{p_τ} and the hash tables $\mathcal{R}$ and $\mathcal{G}$ are returned as output by Function 3 (Line 17).

4 EXPERIMENTAL EVALUATION

In this section, we perform an extensive experimental evaluation to answer the following research questions:

Q1. Why does sampling need deduplication? (Section 4.2)

Q2. How efficient is RadlER to perform deduplicated sampling compared to the traditional batch approach? (Section 4.3)

Q3. What are the benefits of RadlER in terms of runtime and costs compared to the traditional batch approach? (Section 4.4)

Q4. What is the impact of the weighting scheme? (Section 4.5)

[3]The condition is always *True*, unless p_τ was selected as the pivot record in a previous iteration but not inserted into the clean sample $\mathcal{S}$ as it did not match the target group.

Table 1: Features of the selected datasets.

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{D}^c\rvert$ (Avg Size)	$\lvert\mathcal{A}_\mathcal{D}\rvert$	$\mathcal{A}_\mathcal{S}$
alaska_cameras	29.8k	9.0k (3.3)	10	{brand}
nc_voters	14.2k	6.7k (2.1)	14	{sex, race}
nyc_funding	16.3k	3.1k (5.2)	9	{source}
nc_voters_10M	10M	6.6M (1.5)	5	{suburb}

4.1 Experimental Setup

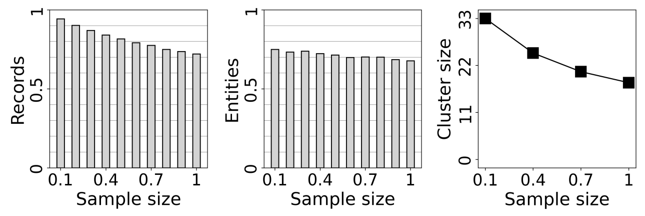
4.1.1 Datasets. We consider three real-world datasets and a synthetic one with heterogeneous features, whose main characteristics are highlighted in Table 1. In particular, the table reports the number of records $|\mathcal{D}|$, the number of described entities $|\mathcal{D}^c|$ with the average number of records describing each of them, the number of attributes $|\mathcal{A}_\mathcal{D}|$, and the default sampling attributes $\mathcal{A}_\mathcal{S}$. All datasets are available in the RadlER GitHub repository.

As part of the Alaska benchmark [15], alaska_cameras is a dataset of $29.8k$ records generated from advertisement about cameras published on e-commerce websites. Since the ground truth is incomplete, we consider the output of a finalist solution [80] from the SIGMOD 2020 Programming Contest [17] as the gold standard. Cameras are partitioned into groups based on their brand. nc_voters[4] contains demographic information about $14.2k$ registered voters from North Carolina [38], which are grouped based on the sex and race attributes. nyc_funding[5] contains $16.3k$ records about financing requests addressed to the NYC Council Discretionary Funding. We perform deduplication on the organizations presenting the requests [19] and create groups based on the source attribute, denoting the type of initiatives. Finally, nc_voters_10M[6] contains $10M$ records generated synthetically from nc_voters [67]. As we are interested in scalability rather than blocking/matching challenges, we consider duplicate records in their original form.

The candidate sets were generated using SparkER [28] meta-blocking techniques for alaska_cameras and nyc_funding, similarity joins [5] for nc_voters, and Soundex [77] for nc_voters_10M. The produced candidate sets are composed of $780k$ record pairs (with a recall of 0.89) for alaska_cameras, $198k$ (0.99) for nc_voters, $231k$ (0.95) for nyc_funding, $27M$ (1) for nc_voters_10M, respectively. Unless otherwise specified, we use the ground truth — acting as an *oracle* — as our default matching function.

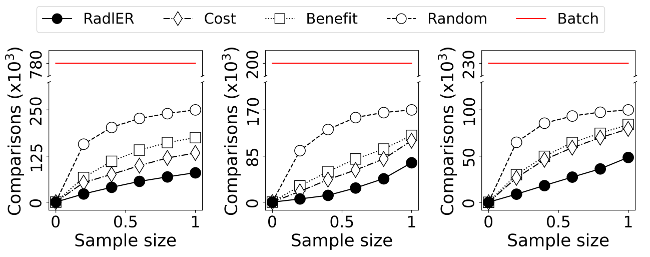
4.1.2 Sampling. Considering the default sampling attributes specified in Table 1, for each dataset we initialized the set of disjoint groups Γ automatically with the most represented groups in the clean version of the dataset, obtained through its ground truth. Further, we set the maximum number of groups to 10 and defined

[4]https://hpi.de/naumann/projects/repeatability/datasets/ncvoters-dataset.html
[5]https://raw.githubusercontent.com/qcri/data_civilizer_system/master/grecord_service/gr/data/address/address.csv
[6]https://dbs.uni-leipzig.de/research/projects/benchmark-datasets-for-entity-resolution



(a) Distinct entities **(b) Partial clusters** **(c) Avg cluster size**

Figure 4: Average measures computed on a random sample of size x produced from alaska_cameras without deduplication.



(a) alaska_cameras **(b) nc_voters** **(c) nyc_funding**

Figure 5: Average number of comparisons performed to produce a clean sample of size x.

a minimum support of 0.01 (i.e., filtering out groups covering less than 1% of the cleaned entities). As the target distribution d, we used equal representation, i.e., $d = [1/|\Gamma|, \forall\, \gamma \in \Gamma]$. The maximum sample size for an undistorted clean sample is equal to about 1100 entities for alaska_cameras, 500 for nc_voters, 400 for nyc_funding, and 912k for nc_voters_10M. Note that we also ran the experiments using demographic parity as the target distribution, reflecting the distribution of the groups in the clean dataset, without spotting significant differences in the number of comparisons and runtime.

4.1.3 Baselines. We compare RadlER against two baselines designed to perform deduplicated sampling. The first baseline, denoted as *batch*, represents the traditional approach requiring to clean the entire data upfront (i.e., to compare all candidates through the matcher), then to sample the produced clean dataset. The second baseline, denoted instead as *random*, represents a naïve solution that we designed to perform deduplicated sampling on-demand. Similarly to RadlER, it maintains a set of records for each group and proceeds iteratively by picking a record to clean from the set of the target group. Nevertheless, the selection is performed randomly, without any weighting scheme. Finally, to assess the impact of the weighting scheme illustrated in Section 3.2, we also take into account two variants of RadlER, each considering only the cost or only the benefit component of the weighting scheme, hence denoted as *cost* and *benefit*, respectively.

4.1.4 Configuration. RadlER has been implemented in Python 3.7. Our experiments were performed on a server equipped with 4 Intel Xeon E5-2697 @ 2.40 GHz (72 cores) processors and 216 GB of RAM, running Ubuntu 18.04.

4.2 Sampling Data with Duplicates

To assess the negative impact of undetected duplicates on sampling, hence the need for deduplication, we provide an intuitive example by performing simple random sampling on alaska_cameras (results are similar for nc_voters and nyc_funding). The measures shown in Figure 4 are computed (averaging over 10 runs) on samples of 10 different sizes ranging from 100 to 1000 records — reported on the x-axis normalized by the maximum sample size.

Figure 4a shows the number of distinct entities described by the records in the sample (normalized by the sample size). This quantity is close to 1 for a sample of 100 records, i.e., there are almost no duplicates among them, but it quickly decreases inversely to the sample size, e.g., a sample of 1000 records represents only ~700 different real-world entities. Figure 4b shows the number of distinct entities that present further duplicates outside the sample (normalized by the number of distinct entities). This quantity is very high (~70%) across all sample sizes. Entities generated from partial clusters of matches can present missing or incorrect values. Further, the presence of duplicates introduces a bias in favor of entities described by more records, whose probability of being represented in the sample is substantially higher. Figure 4c shows the average number of records describing in the original dataset the entities represented in the samples. While this value is on average ~3 considering all dataset entities (as reported in Table 1), it rises up to ~33 for the ones represented in the samples.

A naïve method to obtain a clean sample of size $|\mathcal{S}|$ might be to sample a larger quantity of records (e.g., $2 \times |\mathcal{S}|$), then directly run deduplication on them. However, this approach presents multiple shortcomings. First, as the sampling process was biased (Figure 4c), the distribution of the cleaned entities in the sample would be significantly distorted, over-representing groups whose entities are described by more records in the original data — hence requiring many more comparisons to produce the clean sample. RadlER introduces a weighting scheme precisely to mitigate this bias and enhance efficiency. Further, considering the trend depicted in Figure 4a, it is not even guaranteed that the sample contains enough distinct entities (e.g., a sample of 10k records only describes ~4k distinct entities). Finally, for records in the sample, as most clusters of matches are incomplete (Figure 4b), deduplication should also consider all of their neighbors from the original dataset.

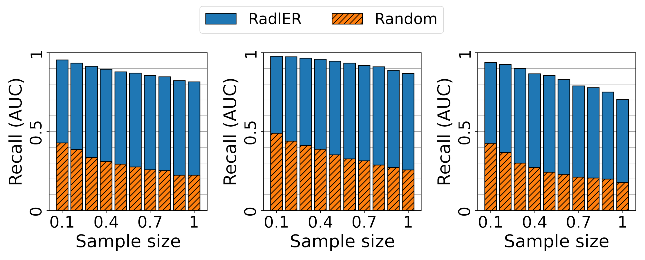
4.3 Performance Evaluation

Figure 5 illustrates the performance of RadlER in terms of required comparisons, i.e., the cost of deduplicated sampling. The lines show the number of comparisons (averaged over 10 runs) required to produce a sample of a given size, considering increasing sizes up to the maximum for an undistorted sample with equal representation (Section 4.1.2) — by which values on the x-axis are normalized.

RadlER significantly outperforms the *batch* approach to deduplicated sampling, which requires performing all comparisons upfront, clearly highlighting the effectiveness of the proposed on-demand approach. For the maximum sample size, the required comparisons are reduced of about 10 times on alaska_cameras (Figure 5a), 3 on nc_voters (Figure 5b), and 5 on nyc_funding (Figure 5c) — the reduction is even greater for smaller sample sizes. Further, RadlER

Table 2: Average runtime and monetary costs to produce a clean sample from alaska_cameras and nc_voters_10M.

	alaska_cameras				nc_voters_10M			
	Size	RadlER	Random	Batch	*Size*	RadlER	Random	Batch
Setup overhead	-	1.18 s	1.18 s	-	-	4.1 m	4.1 m	-
Iteration overhead	108 (0.1)	0.93 s	4.44 s	-	911 (0.001)	26.68 s	47.51 s	-
	540 (0.5)	4.01 s	9.54 s		9.1k (0.01)	4.6 m	7.3 m	
	1080 (1)	6.67 s	11.84 s		91.2k (0.1)	1 h	1.2 h	
Matching runtime with Ditto	108 (0.1)	2.1 m	18.8 m	2.3 h	911 (0.001)	11.08 s	5.6 m	38.2 h
	540 (0.5)	9.1 m	39.4 m		9.1k (0.01)	1.9 m	48.6 m	
	1080 (1)	14.4 m	45.3 m		91.2k (0.1)	20.3 m	3.6 h	
Matching runtime with GPT-4o	108 (0.1)	1.7 h	15 h	110 h	911 (0.001)	18.5 m	9.4 h	159 d
	540 (0.5)	7.3 h	31.6 h		9.1k (0.01)	3.2 h	81 h	
	1080 (1)	11.6 h	36.3 h		91.2k (0.1)	33.7 h	15 d	
Total cost with GPT-4o	108 (0.1)	\$2.89	\$25.49	\$187.25	911 (0.001)	\$0.18	\$5.62	\$2293.07
	540 (0.5)	\$12.37	\$53.56		9.1k (0.01)	\$1.89	\$48.60	
	1080 (1)	\$19.60	\$61.53		91.2k (0.1)	\$20.25	\$217.57	



(a) alaska_cameras **(b) nc_voters** **(c) nyc_funding**

Figure 6: Average progressive recall per comparisons to produce a clean sample of size x.

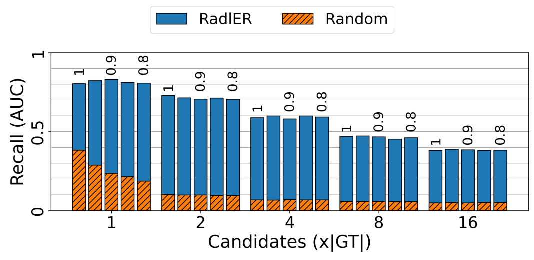


Figure 7: Average progressive recall to produce a clean sample of the maximum size from alaska_cameras with different blocking recall (1, 0.95, 0.9, 0.85, 0.8) and candidate set size.

significantly outperforms even the *random* baseline, a naïve approach to deduplicated sampling on-demand, reducing the number of performed comparisons by more than half. This result highlights the benefits of the weighting scheme described in Section 3.2. Note that both the *cost* and the *benefit* baselines perform better than the *random* one, but are consistently outperformed by RadlER. Thus, even if the two components of the weighting scheme already lead to some advantages taken individually, their combination produces a much more effective reduction in the number of comparisons.

Figure 6 compares RadlER to the *random* baseline in terms of *progressive recall* [27, 46, 71]. For a deduplicated sampling process that generates a clean sample of size $|\mathcal{S}|$, progressive recall at a time t of the process is computed as $|\mathcal{S}_t|/|\mathcal{S}|$, where $|\mathcal{S}_t|$ is the number of cleaned entities inserted into $\mathcal{S}$ by the time t. For each sample size from Figure 5, we produce a line plot with the number of performed comparisons on the x-axis (up to the maximum between RadlER and the *random* baseline for that sample size) and the progressive recall achieved after x comparisons on the y-axis. We compute the (normalized) area under the curve from that plot for both solutions, then represent it through the corresponding column in Figure 6. RadlER requires a much smaller number of comparison to achieve a value y of progressive recall, constantly maintaining an area under the curve at least double that of the *random* baseline.

Finally, for our largest real-world dataset (alaska_cameras) and its maximum sample size, Figure 7 compares RadlER to the *random* baseline in terms of progressive recall on 25 synthetic candidate sets, to analyze the impact of blocking. Candidate sets present different recall (1, 0.95, 0.9, 0.85, 0.8), met by randomly picking a subset of the ground truth, and cardinality (1, 2, 4, 8, and 16 times the ground truth size, i.e., 541k record pairs), obtained by randomly merging clusters of matches to add false positives. Introducing false positives reduces the number of disjoint neighborhoods and increases their average size, hence the number of comparisons. Nevertheless, while progressive recall decreases for both methods over larger candidate sets, RadlER always significantly outperforms the *random* baseline. Further, as highlighted by the first 5 columns, the weighting scheme adopted by RadlER mitigates the bias introduced by false positives that favors records with larger neighborhoods, hence recording more stable performance.

4.4 Runtime and Monetary Cost Analysis

In Table 2, we analyze the impact of RadlER on runtime and monetary costs, comparing it to the *batch* and *random* baselines. For this comparison, we use two datasets: alaska_cameras, our largest real-world dataset and the one with the highest number of tokens per record on average, and nc_voters_10M, to study how RadlER scales over its 10M records and 27M candidates. For every dataset, we consider increasing sample sizes over the maximum for an undistorted sample with equal representation (Section 4.1.2). In particular, in addition to the maximum sample size, we consider 0.1 and 0.5 ratios for alaska_cameras, producing samples ranging between 108 and 1080 entities. For nc_voters_10M, the same ratios would range

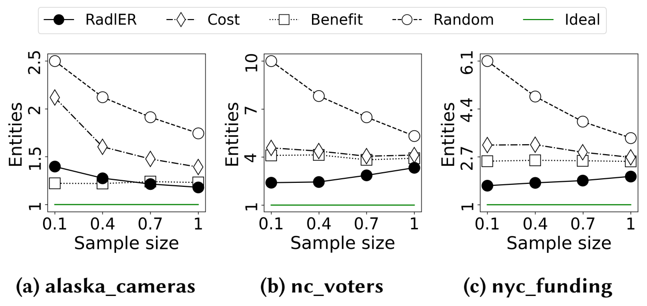


Figure 8: Average number of entities (normalized by the sample size) cleaned to produce a clean sample of size *x*.

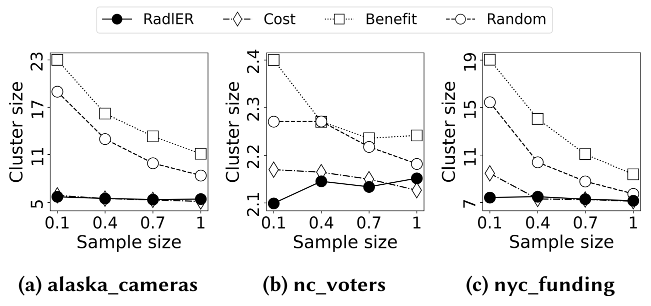


Figure 9: Average number of records describing each entity appearing in a clean sample of size *x*.

between 91.2*k* and 912*k* entities, making results hard to compare and ignoring sample sizes more suitable to the described use cases. Thus, we also cover 0.01 (9.1*k* entities) and 0.001 (911) ratios, reporting the three smallest sample sizes in Table 2 and discussing the cases for 0.5 and 1 ratios in the following.

The first two rows of Table 2 quantify the overhead required by on-demand methods (i.e., RadlER and the naïve *random* baseline) to manage data structures and — for RadlER — weights throughout the setup phase and the subsequent iterations. Then, we report the actual entity matching runtime, estimated for two different matchers: *(i)* Ditto [43], a solution based on pre-trained language models (e.g., BERT [20]) that represented the state-of-the-art before large language models (LLMs) [52], and *(ii)* GPT-4o[7], the most advanced GPT model by OpenAI. Despite their accuracy, LLMs are significantly slower due to their large size and generative nature. In line with our explorative experiments, Peeters et al. [63] report an average runtime per comparison of 0.51 s for GPT-4o in the zero-shot scenario on a product dataset similar to alaska_cameras [62]. For Ditto, we computed instead about 10.6 ms on alaska_cameras and 5.1 ms on nc_voters_10M using an NVIDIA Tesla T4 GPU.

On alaska_cameras, the overhead introduced by on-demand solutions is negligible compared to the entity matching runtime. While RadlER introduces slightly more overhead per iteration than the *random* baseline, due to weighted random selection and weight updates, it requires far fewer iterations, hence its iteration overhead is lower overall. Even with Ditto, the cheapest of the two matchers, RadlER clearly outperforms the *batch* baseline (which requires comparing all candidates), being more than 9 times faster, but also its naïve counterpart, showing at least a x3 improvement. Moving to GPT-4o, RadlER would save almost 25 and 100 hours compared to the *random* and *batch* baselines, respectively.

On nc_voters_10M, the overhead increases significantly, due to the very large amount of records to manage. In the case of Ditto, the iteration overhead contributes more than entity matching to the overall runtime. This trend becomes even more relevant for larger sample sizes, as RadlER records an iteration overhead of 10.1 h and 33.6 h (against an entity matching runtime of 1.7 h and 5.2 h) for 0.5 and 1 ratios (456*k* and 912*k* entities), respectively. In the latter case, on-demand solutions exceed the runtime required by the *batch* baseline. Enhancing the scalability of RadlER on very large sample sizes might represent an interesting challenge for future work. However, for sample sizes reported in Table 2, RadlER consistently outperforms both baselines, reducing the overall runtime by up to 30 and 485 times, respectively.

Regarding the monetary costs introduced by the use of LLMs, using the most updated API pricing by OpenAI[8] and the tiktoken library[9] to compute the average number of tokens per records, we can estimate a cost of 0.024 ¢ (almost 100 tokens per prompt) and 0.0085 ¢ (about 33) per comparison for alaska_cameras and nc_voters_10M with GPT-4o in the zero-shot scenario, respectively. RadlER always leads to a relevant cost reduction, being up to 60 (12*k*) and 8 (30) times cheaper than the *batch* baseline and its naïve counterpart on alaska_cameras (nc_voters_10M). Note that we consider zero-shot since it is the cheapest scenario. Best performing approaches require additional tokens for their prompts [63], hence monetary costs would further increase as well.

4.5 Impact of the Weighting Scheme

As highlighted in Sections 4.3 and 4.4, RadlER clearly outperforms the *random* baseline thanks to its weighting scheme, integrating a cost and a benefit component (Section 3.2). Further, RadlER performs consistently better than the *cost* and *benefit* baselines, which consider each component independently (Figure 5). To provide deeper insights into these results, we analyze how the two components contribute to the achieved improvements.

Figures 8 and 9 report the number of cleaned entities and their average cluster size, respectively — considering the sample sizes introduced in Figure 5. RadlER clearly stands out as the overall best performer, consistently with the results shown in Figure 5. Since the benefit component aims to focus the cleaning effort on entities that are likely to belong to the target group, the *benefit* baseline cleans fewer entities than the *cost* and *random* ones (Figure 8). Nevertheless, as it does not account for neighborhood sizes, it heavily suffers from the bias introduced by duplicates, picking on average the entities with the largest cluster size (Figure 9). This determines many additional comparisons and makes the selection of entities whose actual group is uncertain more frequent. Thus, RadlER shows a further reduction in the number of cleaned entities

[7]https://platform.openai.com/docs/models

[8]https://openai.com/api/pricing/

[9]https://github.com/openai/tiktoken

(up to 5 times compared to the *random* baseline), with the only exception of alaska_cameras (Figure 8a), where sampling attribute values within neighborhoods are generally less heterogeneous.

Since the cost component aims to favor entities that require fewer comparisons, the *cost* baseline consistently reduces the impact of bias (Figure 9). However, its average cluster size is comparable to RadlER, as not accounting for their probability of belonging to the target group results in cleaning many more entities (Figure 8). Finally, the distance of RadlER from the ideal scenario where no entities in excess are cleaned (the line $y = 1$ in Figure 8) leaves room for further optimizations to the current weighting scheme.

5 RELATED WORK

Sampling Data with Duplicates. To the best of our knowledge, very few works study the interaction between deduplication and sampling. SampleClean [78] is an approximate method to apply data cleaning to a small subset of the data, exploiting the outcome to reduce the impact of dirty data on aggregate query answers, e.g., by deduplicating the set of records that impact the most on the average calculation for a certain attribute. Both SampleClean and its extension ActiveClean [39] are approximate methods, and they are not suitable to perform stratified sampling, as they cannot support a user-specified distribution as input.

Finally, Heidari et al. [34] propose an approximate method with error-bound guarantees to sample uniformly at random from the set of entities in the presence of duplicates. That approach relies on locality sensitive hashing [40] to estimate the frequencies of all entities, hence it is not blocking-agnostic. RadlER is an exact method, and it is blocking-agnostic. Thus, it can be seamlessly integrated into existing deduplication pipelines.

Data Cleaning On-Demand. Multiple solutions have been proposed in the existing literature to address the shortcomings of the traditional approach to deduplication, including for instance *progressive* [29, 46, 58, 70, 79] and *query-driven* [1–3] approaches.

In particular, BrewER [71, 81] recently proposed an *on-demand* approach to deduplication, aiming to produce clean data needed by the user incrementally, focusing the cleaning effort on one entity at a time. Nevertheless, the underlying paradigm is substantially different from RadlER. The cleaning process in BrewER is driven by a query issued by the user, consolidating the entities that are likely to appear in the result incrementally, according to a priority defined through the ORDER BY clause. BrewER only supports SP queries, which are not sufficient to perform sampling. On the other hand, RadlER allows defining groups and a target distribution to drive the incremental building of an undistorted clean sample.

Beyond deduplication, on-demand methods for data imputation can be integrated into query execution [11, 44, 64], allowing practitioners to handle missing values dynamically, reducing the query latency and the execution cost. Finally, Giannakopoulou et al. [30] propose a probabilistic repair of denial constraint violations [25] on-demand, driven by SPJ and aggregate queries.

Deduplication & Fairness. In the wake of extensive studies performed in related contexts [49, 66], *bias* and *fairness* recently started to be tackled even from a data-centric perspective. For instance, Shahbazi et al. [69] analyzed the problem of *representation bias* in datasets, while solutions were proposed to define *diversity constraints* in queries [42] or to efficiently generate representative datasets by integrating multiple sources [53].

Nevertheless, only few contributions in the existing literature focus on deduplication. In particular, Shahbazi et al. [68] extensively studied the fairness of the existing matchers. This work is orthogonal to RadlER, which is indeed agnostic towards the choice of the matcher, but it can support the user in the selection of a proper matching function for the task and the data at hand.

FairER [24] introduced the problem of fairness-aware deduplication and suggested an instantiation of that problem for equal representation as the target distribution. FairER partitions the records in a binary fashion through the definition of a *protected group* and acts on the priority of the candidate matches (based by default on the matching probability) to guarantee that protected and non-protected records are equally represented in the result. TREATS [4] extends FairER to a streaming setting, comparing a protected group to multiple other groups, in small batches. Unlike RadlER, FairER and TREATS assume that all candidate matches are compared to compute matching probabilities, similarly to the batch baseline presented here. Finally, RadlER can take into account any type of target distribution, whereas the algorithms presented in FairER and TREATS are specifically targeting equal representation in a record linkage setting (i.e., matching two tables, with each table being duplicate-free), without a data fusion step.

6 CONCLUSION AND FUTURE WORK

This paper provides the first definition of deduplicated sampling as the task of producing a clean sample from a dirty dataset according to a target distribution of entities for some specified groups. We introduced RadlER, a novel on-demand approach to this problem that aims to produce clean samples without cleaning the entire data upfront, focusing instead only on the entities required to appear in the sample. Our experimental evaluation demonstrated that RadlER consistently outperforms the traditional (batch) approach to deduplicated sampling.

In the future, we aim to develop our work in multiple directions. First, we want to study how to further improve the adopted weighting scheme and enhance scalability for very large datasets. Moreover, in addition to bias, we plan to investigate methods that address different data quality dimensions and incorporate them into RadlER. For instance, we plan to investigate whether data fusion for sensitive features may require dedicated functions, as entities belonging to protected groups are more likely to present missing values [47]. Beyond missing value imputation, data quality issues that may need special treatment in terms of data fusion include formatting, noise, and outlier detection, as well as identifying data inconsistencies. We plan to investigate whether variations and extensions of the weighting scheme and the target group selection process can address such additional data quality issues.

ACKNOWLEDGEMENTS

This work was partially supported by Cineca within the ISCRA-C program (code HP10CSFAPN), by MUR within the PRIN "Discount Quality" project (code 202248FWFS), and by HFRI within the "Responsible by Design Entity Resolution" project (GA no. 969).

REFERENCES

[1] Giorgos Alexiou, George Papastefanatos, Vassilis Stamatopoulos, Georgia Koutrika, and Nectarios Koziris. 2025. QueryER: A Framework for Fast Analysis-Aware Deduplication over Dirty Data. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 119–131. https://doi.org/10.48786/edbt.2025.10

[2] Hotham Altwaijry, Dmitri V. Kalashnikov, and Sharad Mehrotra. 2013. Query-Driven Approach to Entity Resolution. *Proceedings of the VLDB Endowment (PVLDB)* 6, 14 (2013), 1846–1857. https://doi.org/10.14778/2556549.2556567

[3] Hotham Altwaijry, Sharad Mehrotra, and Dmitri V. Kalashnikov. 2015. QuERy: A Framework for Integrating Entity Resolution with Query Processing. *Proceedings of the VLDB Endowment (PVLDB)* 9, 3 (2015), 120–131. https://doi.org/10.14778/2850583.2850587

[4] Tiago Brasileiro Araújo, Vasilis Efthymiou, Vassilis Christophides, Evaggelia Pitoura, and Kostas Stefanidis. 2025. TREATS: Fairness-aware entity resolution over streaming data. *Information Systems* 129, Article 102506 (2025), 16 pages. https://doi.org/10.1016/j.is.2024.102506

[5] Nikolaus Augsten and Michael H. Böhlen. 2013. *Similarity Joins in Relational Database Systems*. Springer. https://doi.org/10.1007/978-3-031-01851-0

[6] Nils Barlaug and Jon Atle Gulla. 2021. Neural Networks for Entity Matching: A Survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 3, Article 52 (2021), 37 pages. https://doi.org/10.1145/3442200

[7] Carlo Batini and Monica Scannapieco. 2016. *Data and Information Quality: Dimensions, Principles and Techniques*. Springer. https://doi.org/10.1007/978-3-319-24106-7

[8] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer. https://link.springer.com/book/9780387310732

[9] Jens Bleiholder and Felix Naumann. 2008. Data Fusion. *ACM Computing Surveys* 41, 1, Article 1 (2008), 41 pages. https://doi.org/10.1145/1456650.1456651

[10] Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2024. SC-Block: Supervised Contrastive Blocking Within Entity Resolution Pipelines. In *Proceedings of the European Semantic Web Conference (ESWC), Part 1*. 121–142. https://doi.org/10.1007/978-3-031-60626-7_7

[11] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proceedings of the VLDB Endowment (PVLDB)* 10, 11 (2017), 1310–1321. https://doi.org/10.14778/3137628.3137641

[12] Peter Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer. https://doi.org/10.1007/978-3-642-31164-2

[13] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An Overview of End-to-End Entity Resolution for Big Data. *ACM Computing Surveys* 53, 6, Article 127 (2020), 42 pages. https://doi.org/10.1145/3418896

[14] Sam Corbett-Davies, Johann D. Gaebler, Hamed Nilforoshan, Ravi Shroff, and Sharad Goel. 2023. The Measure and Mismeasure of Fairness. *Journal of Machine Learning Research (JMLR)* 24, Article 312 (2023), 117 pages. http://jmlr.org/papers/v24/22-1511.html

[15] Valter Crescenzi, Andrea De Angelis, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. 2021. Alaska: A Flexible Benchmark for Data Integration Tasks. arXiv:2101.11259

[16] 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 *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1431–1446. https://doi.org/10.1145/3035918.3035960

[17] Andrea De Angelis, Maurizio Mazzei, Federico Piai, Paolo Merialdo, Giovanni Simonini, Luca Zecchini, Sonia Bergamaschi, Donatella Firmani, Xu Chu, Peng Li, and Renzhi Wu. 2023. Experiences and Lessons Learned from the SIGMOD Entity Resolution Programming Contests. *ACM SIGMOD Record* 52, 2 (2023), 43–47. https://doi.org/10.1145/3615952.3615965

[18] Luca Deck, Jan-Laurin Müller, Conradin Braun, Domenique Zipperling, and Niklas Kühl. 2024. Implications of the AI Act for Non-Discrimination Law and Algorithmic Fairness. arXiv:2403.20089

[19] Dong Deng, Wenbo Tao, Ziawasch Abedjan, Ahmed Elmagarmid, Ihab F. Ilyas, Guoliang Li, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Unsupervised String Transformation Learning for Entity Consolidation. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 196–207. https://doi.org/10.1109/ICDE.2019.00026

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), Volume 1 (Long and Short Papers)*. 4171–4186. https://doi.org/10.18653/v1/n19-1423

[21] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration*. Springer. https://doi.org/10.1007/978-3-031-01853-4

[22] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment (PVLDB)* 11, 11 (2018), 1454–1467. https://doi.org/10.14778/3236187.3236198

[23] Vasilis Efthymiou, Ekaterini Ioannou, Manos Karvounis, Manolis Koubarakis, Jakub Maciejewski, Konstantinos Nikoletos, George Papadakis, Dimitris Skoutas, Yannis Velegrakis, and Alexandros Zeakis. 2023. Self-configured Entity Resolution with pyJedAI. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. 339–343. https://doi.org/10.1109/BigData59044.2023.10386556

[24] Vasilis Efthymiou, Kostas Stefanidis, Evaggelia Pitoura, and Vassilis Christophides. 2021. FairER: Entity Resolution With Fairness Constraints. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. 3004–3008. https://doi.org/10.1145/3459637.3482105

[25] Wenfei Fan. 2015. Data Quality: From Theory to Practice. *ACM SIGMOD Record* 44, 3 (2015), 7–18. https://doi.org/10.1145/2854006.2854008

[26] Nikolaos Fanourakis, Vasilis Efthymiou, Vassilis Christophides, Dimitris Kotzinos, Evaggelia Pitoura, and Kostas Stefanidis. 2023. Structural Bias in Knowledge Graphs for the Entity Alignment Task. In *Proceedings of the European Semantic Web Conference (ESWC)*. 72–90. https://doi.org/10.1007/978-3-031-33455-9_5

[27] Donatella Firmani, Barna Saha, and Divesh Srivastava. 2016. Online Entity Resolution Using an Oracle. *Proceedings of the VLDB Endowment (PVLDB)* 9, 5 (2016), 384–395. https://doi.org/10.14778/2876473.2876474

[28] Luca Gagliardelli, Giovanni Simonini, Domenico Beneventano, and Sonia Bergamaschi. 2019. SparkER: Scaling Entity Resolution in Spark. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 602–605. https://doi.org/10.5441/002/edbt.2019.66

[29] Leonardo Gazzarri and Melanie Herschel. 2023. Progressive Entity Resolution over Incremental Data. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 80–91. https://doi.org/10.48786/edbt.2023.07

[30] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning Denial Constraint Violations through Relaxation. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 805–815. https://doi.org/10.1145/3318464.3389775

[31] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-Off Crowdsourcing for Entity Matching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 601–612. https://doi.org/10.1145/2588555.2588576

[32] Isabelle Guyon and André Elisseeff. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research (JMLR)* 3 (2003), 1157–1182. https://jmlr.org/papers/v3/guyon03a.html

[33] Anders Haug, Frederik Zachariassen, and Dennis Van Liempd. 2011. The costs of poor data quality. *Journal of Industrial Engineering and Management (JIEM)* 4, 2 (2011), 168–193. https://doi.org/10.3926/jiem.2011.v4n2.p168-193

[34] Alireza Heidari, Shrinu Kushagra, and Ihab F. Ilyas. 2020. On sampling from data with duplicate records. arXiv:2008.10549

[35] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM. https://doi.org/10.1145/3310205

[36] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI)*. 547–554. https://doi.org/10.1145/2254556.2254659

[37] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 1197–1208. https://doi.org/10.14778/2994509.2994535

[38] Ioannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate Detection with Matching Dependencies. *Proceedings of the VLDB Endowment (PVLDB)* 13, 5 (2020), 712–725. https://doi.org/10.14778/3377369.3377379

[39] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *Proceedings of the VLDB Endowment (PVLDB)* 9, 12 (2016), 948–959. https://doi.org/10.14778/2994509.2994514

[40] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of Massive Datasets*. Cambridge University Press. https://doi.org/10.1017/9781108684163

[41] Guoliang Li. 2017. Human-in-the-loop Data Integration. *Proceedings of the VLDB Endowment (PVLDB)* 10, 12 (2017), 2006–2017. https://doi.org/10.14778/3137765.3137833

[42] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. *Proceedings of the VLDB Endowment (PVLDB)* 17, 2 (2023), 106–118. https://doi.org/10.14778/3626292.3626295

[43] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proceedings*

of the VLDB Endowment (PVLDB) 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

[44] Yiming Lin and Sharad Mehrotra. 2023. ZIP: Lazy Imputation during Query Processing. *Proceedings of the VLDB Endowment (PVLDB)* 17, 1 (2023), 28–40. https://doi.org/10.14778/3617838.3617841

[45] Fabian Lütz. 2024. The AI Act, gender equality and non-discrimination: what role for the AI office? *ERA Forum* 25 (2024), 79–95. https://doi.org/10.1007/s12027-024-00785-w

[46] Jakub Maciejewski, Konstantinos Nikoletos, George Papadakis, and Yannis Velegrakis. 2025. Progressive Entity Matching: A Design Space Exploration. *Proceedings of the ACM on Management of Data (PACMMOD)* 3, 1, Article 65 (2025), 25 pages. https://doi.org/10.1145/3709715

[47] Fernando Martínez-Plumed, Cèsar Ferri, David Nieves, and José Hernández-Orallo. 2021. Missing the missing values: The ugly duckling of fairness in machine learning. *International Journal of Intelligent Systems* 36, 7 (2021), 3217–3258. https://doi.org/10.1002/int.22415

[48] Vamsi Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1133–1147. https://doi.org/10.1145/3318464.3380597

[49] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2022. A Survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys* 54, 6, Article 115 (2022), 35 pages. https://doi.org/10.1145/3457607

[50] Sandy Moens and Bart Goethals. 2013. Randomly Sampling Maximal Itemsets. In *Proceedings of the Workshop on Interactive Data Exploration and Analytics (IDEA @ KDD)*. 79–86. https://doi.org/10.1145/2501511.2501523

[51] 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 *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 19–34. https://doi.org/10.1145/3183713.3196926

[52] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment (PVLDB)* 16, 4 (2022), 738–746. https://doi.org/10.14778/3574245.3574258

[53] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2021. Tailoring Data Source Distributions for Fairness-aware Data Integration. *Proceedings of the VLDB Endowment (PVLDB)* 14, 11 (2021), 2519–2532. https://doi.org/10.14778/3476249.3476299

[54] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection*. Springer. https://doi.org/10.1007/978-3-031-01835-0

[55] Matteo Paganelli, Francesco Del Buono, Andrea Baraldi, and Francesco Guerra. 2022. Analyzing How BERT Performs Entity Matching. *Proceedings of the VLDB Endowment (PVLDB)* 15, 8 (2022), 1726–1738. https://doi.org/10.14778/3529337.3529356

[56] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. 2014. Meta-Blocking: Taking Entity Resolution to the Next Level. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26, 8 (2014), 1946–1960. https://doi.org/10.1109/TKDE.2013.54

[57] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Computing Surveys* 53, 2, Article 31 (2020), 42 pages. https://doi.org/10.1145/3377455

[58] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. 2015. Progressive Duplicate Detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 27, 5 (2015), 1316–1329. https://doi.org/10.1109/TKDE.2014.2359666

[59] Eliana Pastor, Luca de Alfaro, and Elena Baralis. 2021. Looking for Trouble: Analyzing Classifier Behavior via Pattern Divergence. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1400–1412. https://doi.org/10.1145/3448016.3457284

[60] Derek Paulsen, Yash Govind, and AnHai Doan. 2023. Sparkly: A Simple yet Surprisingly Strong TF/IDF Blocker for Entity Matching. *Proceedings of the VLDB Endowment (PVLDB)* 16, 6 (2023), 1507–1519. https://doi.org/10.14778/3583140.3583163

[61] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for Entity Matching. In *Proceedings of the European Conference on Advances in Databases and Information Systems (ADBIS): Short Papers, Doctoral Consortium and Workshops*. 221–230. https://doi.org/10.1007/978-3-031-42941-5_20

[62] Ralph Peeters, Reng Chiz Der, and Christian Bizer. 2024. WDC Products: A Multi-Dimensional Entity Matching Benchmark. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 22–33. https://doi.org/10.48786/edbt.2024.03

[63] Ralph Peeters, Aaron Steiner, and Christian Bizer. 2023. Entity Matching using Large Language Models. arXiv:2310.11244

[64] Massimo Perini and Milos Nikolic. 2024. In-Database Data Imputation. *Proceedings of the ACM on Management of Data (PACMMOD)* 2, 1, Article 70 (2024), 27 pages. https://doi.org/10.1145/3639326

[65] Dana Pessach and Erez Shmueli. 2023. A Review on Fairness in Machine Learning. *ACM Computing Surveys* 55, 3, Article 51 (2023), 44 pages. https://doi.org/10.1145/3494672

[66] Evaggelia Pitoura, Kostas Stefanidis, and Georgia Koutrika. 2022. Fairness in rankings and recommendations: an overview. *VLDB Journal* 31, 3 (2022), 431–458. https://doi.org/10.1007/S00778-021-00697-y

[67] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2017. Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution. In *Proceedings of the European Conference on Advances in Databases and Information Systems (ADBIS)*. 278–293. https://doi.org/10.1007/978-3-319-66917-5_19

[68] Nima Shahbazi, Nikola Danevski, Fatemeh Nargesian, Abolfazl Asudeh, and Divesh Srivastava. 2023. Through the Fairness Lens: Experimental Analysis and Evaluation of Entity Matching. *Proceedings of the VLDB Endowment (PVLDB)* 16, 11 (2023), 3279–3292. https://doi.org/10.14778/3611479.3611525

[69] Nima Shahbazi, Yin Lin, Abolfazl Asudeh, and H. V. Jagadish. 2023. Representation Bias in Data: A Survey on Identification and Resolution Techniques. *ACM Computing Surveys* 55, 13, Article 293 (2023), 39 pages. https://doi.org/10.1145/3588433

[70] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2018. Schema-agnostic Progressive Entity Resolution. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 53–64. https://doi.org/10.1109/ICDE.2018.00015

[71] Giovanni Simonini, Luca Zecchini, Sonia Bergamaschi, and Felix Naumann. 2022. Entity Resolution On-Demand. *Proceedings of the VLDB Endowment (PVLDB)* 15, 7 (2022), 1506–1518. https://doi.org/10.14778/3523210.3523226

[72] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *Proceedings of the VLDB Endowment (PVLDB)* 11, 2 (2017), 189–202. https://doi.org/10.14778/3149193.3149199

[73] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*. 3645–3650. https://doi.org/10.18653/v1/P19-1355

[74] 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. *Proceedings of the VLDB Endowment (PVLDB)* 14, 11 (2021), 2459–2472. https://doi.org/10.14778/3476249.3476294

[75] Steven K. Thompson. 2012. *Sampling*. John Wiley & Sons. https://doi.org/10.1002/9781118162934

[76] Tània Verge. 2010. Gendering Representation in Spain: Opportunities and Limits of Gender Quotas. *Journal of Women, Politics & Policy* 31, 2 (2010), 166–190. https://doi.org/10.1080/15544771003697247

[77] Valery S. Vykhovanets, Jianming Du, and Sergey A. Sakulin. 2020. An Overview of Phonetic Encoding Algorithms. *Automation and Remote Control* 81 (2020), 1896–1910. https://doi.org/10.1134/S0005117920100082

[78] Jiannan Wang, Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. 2014. A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 469–480. https://doi.org/10.1145/2588555.2610505

[79] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. 2013. Pay-As-You-Go Entity Resolution. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 25, 5 (2013), 1111–1124. https://doi.org/10.1109/TKDE.2012.43

[80] Luca Zecchini, Giovanni Simonini, and Sonia Bergamaschi. 2020. Entity Resolution on Camera Records without Machine Learning. In *Proceedings of the International Workshop on Challenges and Experiences from Data Integration to Knowledge Graphs (DI2KG @ VLDB)*. https://ceur-ws.org/Vol-2726/paper3.pdf

[81] Luca Zecchini, Giovanni Simonini, Sonia Bergamaschi, and Felix Naumann. 2023. BrewER: Entity Resolution On-Demand. *Proceedings of the VLDB Endowment (PVLDB)* 16, 12 (2023), 4026–4029. https://doi.org/10.14778/3611540.3611612

[82] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. FA*IR: A Fair Top-k Ranking Algorithm. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. 1569–1578. https://doi.org/10.1145/3132847.3132938