Sudowoodo: Contrastive Self-supervised Learning for Multi-purpose Data Integration and Preparation

Runhui Wang§
Computer Science Department
Rutgers University
New Brunswick, United States
runhui.wang@rutgers.edu

Yuliang Li
Megagon Labs
Mountain View, United States
yuliang@megagon.ai

Jin Wang
Megagon Labs
Mountain View, United States
jin@megagon.ai

Abstract—Machine learning (ML) is playing an increasingly important role in data management tasks, particularly in Data Integration and Preparation (DI&P). The success of ML-based approaches, however, heavily relies on the availability of large-scale, high-quality labeled datasets for different tasks. Moreover, the wide variety of DI&P tasks and pipelines oftentimes requires customizing ML solutions at a significant cost for model engineering and experimentation. These factors inevitably hold back the adoption of ML-based approaches to new domains and tasks.

In this paper, we propose Sudowoodo, a multi-purpose DI&P framework based on contrastive representation learning. Sudowoodo features a unified, matching-based problem definition capturing a wide range of DI&P tasks including Entity Matching (EM) in data integration, error correction in data cleaning, semantic type detection in data discovery, and more. Contrastive learning enables Sudowoodo to learn similarity-aware data representations from a large corpus of data items (e.g., entity entries, table columns) without using any labels. The learned representations can later be either directly used or facilitate fine-tuning with only a few labels to support different DI&P tasks. Our experiment results show that Sudowoodo achieves multiple state-of-the-art results on different levels of supervision and outperforms previous best specialized blocking or matching solutions for EM. Sudowoodo also achieves promising results in data cleaning and column matching tasks showing its versatility in DI&P applications.

I. INTRODUCTION

Machine learning, particularly deep learning (DL), is playing an increasingly important role in almost all fields in computer science including data management [1], [2], [3], [4]. More recently, this trend has extended to Data Integration and Preparation (DI&P) [5], [6], [7], [8] with learning-based solutions making promising progress and advancing the state-of-the-art in multiple tasks, including entity matching [9], [10], data cleaning [11], [12], [13], [14], and table annotation [15], [16], [17]. For example, Entity Matching (EM) solutions based on pre-trained language models such as BERT [18] achieve new state-of-the-art results across multiple matching tasks [19], [10]. While shown to be effective, as we describe next, existing learning-based solutions for DI&P suffer from two major challenges making them less attractive in practice.

Challenge 1: Label Requirement. The success of ML-based approaches comes at the cost of creating large-scale, high-quality annotated datasets. For example, in Entity Matching, learning-based methods typically require around 10k or more labeled entity pairs to achieve the best pairwise matching performance [9], [10]. Creating high-quality datasets of such cardinality can be quite expensive considering the required human efforts, which make ML-based solutions less accessible to new DI&P tasks and domains.

Challenge 2: Task Variety. There is a wide scope of DI&P tasks requiring ML models for different problem formulations. This means that oftentimes, practitioners have to build a specialized ML solution for each task, resulting in extra costs of model engineering. For example, Entity Matching typically consists of two tasks, blocking and matching [20], with the goal of (1) filtering out candidate matching pairs and (2) performing the pairwise comparison, respectively. The two tasks are closely related, but because of their different problem formulations, existing studies typically propose specialized solutions [21], [9], [19], [22], [10] that require separate modeling, data annotation, and experimentation efforts. for each task. This has also been the case for other DI&P tasks in data cleaning and data discovery [11], [12], [13], [14], [15], [16], [17].

To address these two challenges, we propose Sudowoodo, a multi-purpose DI&P framework based on contrastive representation learning [23]. Sudowoodo features a generic DI&P problem definition of data matching: given a collection of data items (e.g., entity entries, table columns, cell values), identify all pairs of compatible items. This generic definition allows Sudowoodo to support a wide range of tasks in data integration, cleaning, and discovery. As Figure 1 illustrates, EM can be formulated as a task of matching entity entries; error correction in data cleaning [12], [13], [14] can be formulated as matching erroneous cell values with candidate corrections; and column matching, which is widely used in data discovery [24], [15], can be formulated as matching table columns that have the same semantic type. The matching criteria are also customizable w.r.t the specific application. In this work, we consider any criteria that can be cast into a general binary relation, such as whether two entity entries refer to the same real-world entity for EM.

Sudowoodo addresses the label requirement challenge by
Fig. 1: Sudowoodo supports any DI&P tasks that can be formulated as a general form of matching task.

leveraging contrastive learning to learn a data representation model from a large collection of unlabeled data items, such as corpora of entity entries or table columns. Unlike that of language models such as BERT [18], the effectiveness of Sudowoodo’s pre-training process relies on its contrastive learning objective instead of learning linguistic characteristics. The contrastive objective allows Sudowoodo to learn how to distinguish pairs of similar data items from dissimilar ones that are likely to be distinct. In this way, without using any labels, Sudowoodo learns meaningful representations where similar data items are close in the representation space while pairs of dissimilar data items are far apart.

After the pre-training process, the learned data representation model is accessible to downstream tasks either directly in an unsupervised manner or by fine-tuning using a few labels. Unlike that in language models, fine-tuning is not always necessary if the downstream task is similarity-based for which Sudowoodo already provides a reliable metrics space. When fine-tuning is needed, Sudowoodo provides an optional pseudo labeling step that extracts training signals with high confidence from the learned representations useful for further boosting the fine-tuning performance. As we will illustrate, pseudo labeling together with a customized fine-tuning architecture significantly improves Sudowoodo’s performance on DI&P tasks in low-resource settings.

Example 1: We illustrate the Sudowoodo pipeline in Figure 2 using Entity Matching (EM), a typical application of data integration. The upper part of the figure shows a standard EM pipeline consisting of the blocking and matching sub-tasks. Given two collections of entity entries (Tables A and B), the goal of blocking is to select a candidate set of matching pairs from the cross product to avoid the quadratic-size pairwise comparison. Next, it trains a matching model on a labeled dataset sampled from the candidate set, before applying the model to all the candidate pairs for the final matching results.

Sudowoodo starts with contrastive learning to learn the similarity-aware data representations of all entity entries from the two input tables. Because its model already captures entity similarity, we can solve the blocking task by applying nearest neighbor search to identify the top-k most similar entity entries for all input entries as the candidate set. For the matching task, practitioners can then fine-tune the pre-trained representation model on a labeled training set. For this step, Sudowoodo’s pseudo labeling module allows automatic creation of additional high-quality labels by extracting match/unmatch pairs with high confidence from the learned representations. By doing so, Sudowoodo significantly reduces the label requirement for training the matching model.

Further optimizations. Based on the standard contrastive learning framework, Sudowoodo features several optimizations to further boost the pre-training effectiveness. First, the quality of learned representations heavily depends on the generated pairs of data items that are semantic-preserving. Sudowoodo obtains such pairs via data transformation such as synonym replacement or span deletion. To obtain more fine-grained transformations, Sudowoodo supports the cutoff data augmentation (DA) technique [25] that transforms input data items at the word-embedding level. Combining cutoff with a package of DI&P data augmentation operators, Sudowoodo obtains more diverse views for contrastive pre-training.

Another challenge for applying contrastive learning to DI&P tasks is how to obtain more effective negative examples. Negative examples allow the representation model to learn how to separate dissimilar data items. The standard contrastive learning approach uses uniformly sampled pairs as negative examples. This simple approach can be insufficient when random pairs fail to provide strong enough negative examples because they are easily separable. To address this issue, Sudowoodo splits the unlabeled data items into clusters and samples negative pairs within each cluster. By doing so, Sudowoodo allows the representation model to learn “harder” by distinguishing syntactically similar items.

In summary, this paper makes the following contributions:

- We propose a generic matching problem definition that captures a wide variety of DI&P tasks including (1) EM in data integration, (2) Error correction in data cleaning, (3) Semantic type detection in data discovery, and more.
- We propose Sudowoodo, a contrastive representation learning framework for the supported DI&P tasks. Sudowoodo pre-trains a representation model from a large collection of unlabeled data items. Practitioners can then apply the representations to downstream tasks either directly or by
fine-tuning with few labels.

- We introduce 3 optimizations to Sudowoodo. Specifically, we (1) augment the pre-training data using a novel cutoff operator, (2) introduce “harder” negative samples via clustering, and (3) combine the standard contrastive loss with a non-contrastive learning objective through redundancy regularization [26].
- We evaluate Sudowoodo on real-world datasets for EM, data cleaning, and semantic type detection. Sudowoodo achieves a series of new SOTA results under different label budgets, highlighting its capability of holistically providing solutions for multiple DI&P tasks.

The rest of the paper is organized as follows. Section II defines the problem and introduces the overall architecture. We introduce the contrastive learning algorithms in Section III and their optimizations in Section IV. Section V extends Sudowoodo to data cleaning and column matching. Section VI summarizes the experiment results. We discuss related work in Section VII and conclude in Section VIII.

II. BACKGROUND AND SYSTEM OVERVIEW

In this section, we first define a general matching task for capturing a broad set of DI&P tasks including Entity Matching (EM), data cleaning, and column type detection. We then review the baseline method of pre-trained language models and fine-tuning for solving the task. We also provide an overview of Sudowoodo’s architecture.

A. Task definition

In Sudowoodo, we consider DI&P tasks that can be formulated as a general problem of matching related data items. These data items can be of diverse formats across different DI&P tasks. For example, as Figure 1 illustrates, these data items can be entity entries, table columns, cell values, etc. The matching criteria can also be customizable binary relations w.r.t the specific DI&P applications, such as: whether two entity entries refer to the same real-world entity for EM; whether a row is a correct candidate correction to an erroneous row for data cleaning; or whether two table columns have the same semantic type for data discovery.

It is commonly seen that within a DI&P task, the same matching problem is formulated in different ways to address different application needs. For example, in an EM task, the blocking stage requires quickly getting a small candidate set that contains filtered matching pairs with high recall while the matching stage requires both high recall and precision. To facilitate their different needs, in Sudowoodo, we consider two flavors of the matching tasks: embedding and pairwise matching. The embedding task aims at generating a vector representation for each data item so that related data items are close in the vector space. The generated vectors can then be indexed and queried to support fast similarity search [27] or clustering [28]. The goal of the pairwise matching task is to perform binary classification for determining whether a pair of data items are related or not. We formally define these two tasks as follows:

**Definition 1 (Embedding):** Given a collection \( D \) of data items, a \( d \)-dimensional embedding model \( M_{\text{emb}} \) takes every data item \( x \in D \) as input and outputs a real vector \( M_{\text{emb}}(x) \in \mathbb{R}^d \). A similarity function \( \text{sim} \), e.g., cosine, for every pair of data items \((x, x')\), the value of \( \text{sim}(x, x') \) is large if and only if \((x, x')\) matches.

For simplicity, we assume all output vectors are normalized, i.e. the L-2 norm \( \| M_{\text{emb}}(x) \|_2 = 1 \) for all data item \( x \).

**Definition 2 (Pairwise Matching):** Given a collection \( D \) of data items, a pairwise matching model \( M_{\text{pm}} \) takes a pair \((x, y)\) of data items as input and outputs \( M_{\text{pm}}(x, y) \in \{0, 1\} \) where \( M_{\text{pm}}(x, y) = 1 \) if \((x, y)\) is a real match or 0 otherwise.

Note that we can also abuse the notation \( M_{\text{pm}}(x, y) \) as a 2-d real vector where each dimension indicates the predicted probability of non-match / match denoted as \( M_{\text{pm}}(x, y)_0 \) and \( M_{\text{pm}}(x, y)_1 \) respectively.

**Entity Matching.** With the help of Sudowoodo, it is rather easy to build an effective EM pipeline. The embedding model provides the similarity measure for the blocking stage of EM while the matching stage can directly use the pairwise matching model that Sudowoodo provides. For the rest of the paper, we will take EM as the main example when introducing the Sudowoodo framework for clarity. We will describe how to extend Sudowoodo to the data cleaning and column type detection tasks later in Section V.

B. Fine-tuning language models

The goal of Sudowoodo is to (1) train high-quality embedding models without labels and (2) train the pairwise matching models using no or just a few labels. Pre-trained language models (LMs), such as BERT, provide a good baseline solution which has recently achieved promising results for EM [10], [29], [22], [30]. These models typically consist of 12 or more Transformer layers [31] pre-trained on large unlabeled text corpora such as Wikipedia. In the process of pre-training, the LMs are trained in a self-supervised manner to perform auxiliary tasks such as missing token and next-sentence prediction. In this way, LMs effectively capture lexical or semantic meanings of the input text sequences. Formally, a LM \( M \) encodes an input token sequence \( x \) into a vector representation \( M(x) \).

To apply pre-trained LMs in Sudowoodo, we first need to serialize the input data items into sequences of tokens. For EM, we follow the serialization method in Ditto [10] which converts an entity entry into the format:

\[
\text{[COL] Title [VAL] instant immers... \[COL] Price [VAL] 36.11}
\]

where [COL] and [VAL] are special tokens indicating the start of an attribute or value. We denote by \( \text{serialize}(x) \) the serialization of a data item \( x \). We omit “serialize” when the context is clear. After that, we can obtain a baseline embedding model that first serializes an input data item and encodes it using the pre-trained LM.

Next, one can obtain a baseline pairwise matching model via fine-tuning the pre-trained model on a downstream task-specific training set. Pre-trained LMs like BERT support...
sequence pair classification by concatenating the input pair \((x, y)\) into a single sequence as

\[
[CLS] \text{serialize}(x) [SEP] \text{serialize}(y) [SEP]
\]

where the [SEP]'s are separator tokens and [CLS] is a special token to obtain the sequence pair representation. Fine-tuning typically consists of the following steps:

- Add task-specific layers after the last Transformer layer. In our case, the task-specific layers consist of a linear fully connected layer and a softmax layer for binary classification.
- Initialize the modified network with the pre-trained weights.
- Train the modified network until convergence.

Since the LM is pre-trained to gain semantic understanding capability, it is expected that the fine-tuning approach requires fewer labels than training the model entirely from scratch.

C. Overview of Sudowoodo

The pre-trained LM approach works reasonably well, but it is sub-optimal in the DI&P tasks supported by Sudowoodo. The main reason is that the pre-training tasks do not train the LM to explicitly capture the similarity between data items. For the embedding model (i.e., blocker), without further fine-tuning [32], [16], [22], the LM representations for two closely related data items may not be sufficiently close to support effective similarity search. The fine-tuning process of concatenating input data items also does not model the “differences” between data items potentially leading to lower matching accuracy or a higher label requirement.

To address this challenge, Sudowoodo utilizes the contrastive learning [23] technique. Unlike LM pre-training, contrastive learning explicitly captures relatedness between data items. The pre-training objective is designed to enforce vector representations of similar data items to be close and different items to be far apart. We will introduce the contrastive learning algorithm in Section III and its optimizations in Section IV.

The lower part of Figure 2 illustrates the overall framework of Sudowoodo in the EM application. Sudowoodo starts with a pre-training phase ① with contrastive learning. During pre-training, Sudowoodo continuously updates a pre-trained LM using positive/negative pairs of data items created by data transformation and negative sampling methods. This process produces an embedding model more suitable to be used in step ②. For blocking, we apply the embedding model to vectorize each data item and use high-dimensional similarity search technique to filter out the \(k\) nearest items for every input as the candidate set.

To further reduce the label requirement of fine-tuning, in step ③, Sudowoodo applies pseudo labeling techniques (Section III-C) to automatically generate probabilistic training examples. These examples are combined with the manually labeled ones to form the training set for the fine-tuning step ④. For fine-tuning, Sudowoodo features a new task-specific layer that captures the difference between data representations beyond simple concatenation.

Discussion about scalability. The design of the Sudowoodo pipeline is optimized towards delivering high-quality matching results. In practice, one might train and deploy the models in different manners to achieve a good balance between quality and scalability, which is beyond the scope of this work. However, we notice that by utilizing Sudowoodo’s learned representations, the blocking stage tends to produce a significantly smaller candidate set with high recall that leads to acceleration of the matching stage.

III. CONTRASTIVE LEARNING

We overview contrastive learning in Section III-A and explain why directly applying the SimCLR algorithm is not sufficient for Sudowoodo’s matching problem settings. We propose two novel techniques to address the issue: (1) a customized fine-tuning approach and (2) pseudo-labeling in Section III-B and III-C respectively.

A. Representation learning with SimCLR

Contrastive learning is a self-supervision approach that learns data representations where similar data items are close while different ones are far apart. In Sudowoodo, we choose SimCLR [23], which has shown effectiveness in learning visual representations and NLP tasks [33], as the base algorithm. At a high level, SimCLR pre-trains a representation model by simultaneously (1) minimizing distance between pairs of same/similar data items and (2) maximizing distance between pairs of distinct items as Figure 3 illustrates.

To achieve this goal without labels, for objective (1), we utilize data augmentation operators to generate variants (i.e., multiple views) of the same data item. These operators are task-specific data transformations that preserve the semantics of the input data item. For example, in EM, such transformations can be permuting the attribute order or replacing a token with its synonym.

Minimizing objective (1) only can result in a trivial solution where all representations collapse into a single vector. SimCLR leverages negative sampling to address this issue in objective (2). The default negative sampling method uses uniformly sampled pairs as negative pairs. This is based on the observation that two randomly sampled items are likely to be distinct in a large corpus.

We outline the pseudo code of SimCLR in Algorithm 1. The algorithm consists of four major steps:

- Data augmentation (DA, Line 7). This module contains a set of DA operators that generate semantically equivalent distorted views \((B_{ori} \text{ and } B_{aug})\) of the same data item.
The main reason is that after pre-training, the performance. The main reason is that after pre-training, the
model captures representations of single data items, while the downstream tasks of DI&P need to handle pairs of data items as input. Without special treatment, the default fine-tuning option cannot make full use of the rich semantics captured during pre-training. Next, we introduce two novel techniques to address this challenge.

B. Fine-tuning in Sudowoodo

After pre-training, the embedding model captures similarity among data items. For pairwise matching, the default option of LM fine-tuning is to first concatenate the input pair of serialized data items and then encode the concatenated sequence using $M_{\text{emb}}$. This is not ideal because $M_{\text{emb}}$ is pre-trained to encode single items instead of concatenated pairs. On the other hand, encoding the two data items separately can miss cross entity information (e.g., the difference between two product IDs), which can be captured by the LM’s self-attention mechanism applied to the concatenated pairs.

In Sudowoodo, we extend the default fine-tuning with an extra step for capturing similarity/difference between the input pairs. We illustrate the fine-tuning model architecture in Figure 4. Specifically, given a pair $(x, y)$ of serialized data items, we applied the pre-trained representation model $M_{\text{emb}}$ on $x$ and $y$ individually and on the concatenation $xy = \text{concat}(x, y)$. Let $Z_x$, $Z_y$ and $Z_{xy}$ be the $d$-dimensional representations $M_{\text{emb}}(x)$, $M_{\text{emb}}(y)$, and $M_{\text{emb}}(xy)$. The task-specific layers of Sudowoodo consist of a linear layer Linear$\text{diff}$ of input dimension $2d$ and a softmax function that predicts whether the pair matches or not (0/1). The pairwise matching model $M_{\text{pm}}$ is defined as

$$M_{\text{pm}}(x, y) = \text{softmax}(\text{Linear}_\text{diff}(Z_{xy} \oplus |Z_x - Z_y|))$$

where $\oplus$ denotes vector concatenation. Computing the vector subtraction and absolute values is done element-wise.

C. Pseudo labeling

In addition to fine-tuning, Sudowoodo also utilizes a pseudo labeling step to extract similarity-based knowledge from the learned data item to further boost the performance of pairwise matching problems. This is realized by automatically generating probabilistic labels to augment the manually labeled set. It works for pairwise matching tasks that are similarity-based, for which the embedding model $M_{\text{emb}}$ provides a reliable resource for testing whether a pair of data items are similar or not.

At a high level, for every unlabeled pair $(x, y)$ in the candidate set, Sudowoodo measures the confidence of whether $x$ and $y$ matches by the cosine similarity between their representations $M_{\text{emb}}(x)$ and $M_{\text{emb}}(y)$. We assign the pair the positive label if the cosine similarity $\cos(M_{\text{emb}}(x), M_{\text{emb}}(y))$ is above a threshold.
is above a positive threshold $\theta^+$ and the negative label if their similarity is below a negative threshold $\theta^-$. The choices for hyper-parameters $\theta^+$ and $\theta^-$ are crucial as they determine the quality and size of the pseudo labels. For example, a more relaxed combination of $(\theta^+, \theta^-)$ can result in a training set of larger size but more noise in the labels.

Yet another important factor to consider is balancing the ratio between positive and negative labels. DI&P tasks such as EM or data cleaning naturally have much more negative labels (i.e., non-match pairs / clean cells) than positive ones. A training set with a high ratio of negative pairs can result in a matching model that is heavily biased towards predicting negative during fine-tuning.

In Sudowoodo, we take a semi-automatic approach to tune $(\theta^+, \theta^-)$ as hyper-parameters. The user first needs to fix a positive ratio $\rho$ from a small set $\{5\%, 10\%, \ldots \}$. This ratio can also be estimated using a few uniformly sampled labels. Let $C$ be the candidate set and $C^+ \cup C^-$ be the subset of $C$ above $\theta^+$ or below $\theta^-$ respectively. By fixing $\rho$, we would like to keep $|C^+|/(|C^+| + |C^-|) = \rho$ and only search for $\theta^+$ (or $\theta^-$). We then take a hill-climbing heuristics [34] to find a locally optimal $\theta^+$ using a fixed number of fine-tuning trials.

IV. PRE-TRAINING OPTIMIZATIONS

By the design of the contrastive learning framework, the quality of the representations depends on the quality of (1) data augmentation (DA) operators for generating positive pairs and (2) negative sampling for generating distinctive pairs. For (1), we propose a task-specific DA method with cutoff [25] which perturbs the input token embeddings of the LM directly to generate augmentations with more diversity. For (2), we propose a clustering-based sampling technique that samples negative pairs from the same cluster. We expect the negative samples to become “harder” to distinguish thus the representations model is forced to learn more meaningful features.

In Sudowoodo, we also improve the pre-trained model by combining contrastive learning with Barlow Twins [26], a recently proposed self-supervised learning technique based on feature redundancy regularization. Without relying on negative sampling, redundancy regularization enforces the learned features in the representation space to be independent of each other so that it avoids having multiple features capturing the same entity information.

A. Data augmentation (DA) with cutoff

The original goal of DA is to generate additional training examples from existing ones. DA operators are typically data transformations that preserve the semantic meaning of an input data item. In computer vision, these transformations can be rotation, flipping, or cropping of an image. DA has also been applied to learning-based DI&P tasks (see [35] for a survey). In contrastive learning, DA has a slightly different goal of generating pairs of data items that are similar to each other. It is, therefore, less of an issue of getting corrupted labels (e.g., identical data items becoming distinct) as long as the transformation preserves a certain level of data similarity.

### TABLE I: DA operators in Sudowoodo for EM.

| Operators   | Details                                      |
|-------------|----------------------------------------------|
| token_del   | Sample and delete a token                    |
| token_repl  | Sample a token and replace it with a synonym  |
| token_swap  | Sample two tokens and swap them              |
| token_insert| Sample a token and insert to its right a synonym |
| span_del    | Sample and delete a span of tokens           |
| span_shuffle| Sample a span of tokens and shuffle their order |
| col_shuffle | Choose two attributes and swap their order    |
| col_del     | Choose an entity attribute and drop it entirely |

| Operators Details | Ratios |
|-------------------|--------|
| token_del         | 75 97 88 16 65 |
| token_repl        | 39 92 78 75 8 |
| token_swap        | 82 82 58 25 80 |
| token_insert      | 92 94 9 64 89 |
| span_del          | 9 90 58 7 45  |
| span_shuffle      | 9 90 0 7 45   |
| col_shuffle       | 9 0 0 0 7 45  |
| col_del           | 75 97 0 16 65 |

Fig. 5: Three cutoff DA operators. The cutoff operators modify rows/columns of the input token embedding matrices directly.

Based on this intuition, we design the DA operators in Sudowoodo by combining a base set of task-specific operators with additional perturbation using the cutoff operator [25]. Following previous works [10], [36], we support the set of DA transformation operators listed in Table I for EM.

The supported operators include token-based, span-based, and attribute-based transformations. In this work, we apply a single base DA operator at a time. We note that one might also apply a learning-based DA such as Rotom [36] to automatically select and combine multiple DA operators, which we leave as future work.

Sudowoodo then combines the base transformations with a cutoff operator. The cutoff operators directly modify the input token embeddings to the representation model. We consider 3 cutoff operators: token-cutoff, feature-cutoff, and span-cutoff. As Figure 5 illustrates, given an input serialized data item, after converted to a sequence of token embeddings (each column is the embedding of an input token), these 3 operators uniformly sample a token index, a set of feature indices, or a span of token indices and set the corresponding dimensions in the embedding matrix to zeros.

Sudowoodo performs the cutoff operators batch-wise, i.e., the same token/feature/span-cutoff is applied to all data items in the same batch. This allows the cutoff to gain additional effects that at each training step, the Transformer-based encoder uses only partial information to make the matching prediction. It prevents the encoder to “overfit” certain features or parts of the input content. A similar design idea is used in the popular dropout mechanism [37] for training deep neural networks.

B. Clustering-based negative sampling

As mentioned in Section III, the default sampling method uses pairs of uniformly sampled data items as negative examples. This method can be insufficient because it can be quite easy to differentiate such pairs, e.g., by checking if the data items contain overlapping tokens, without capturing the more important feature such as product ID’s.
and first applies DA operators to generate two different views Figure 6 illustrates. Similar to SimCLR, the BT algorithm BT has an identical structure with the SimCLR algorithm as ages redundancy regularization. The pre-training algorithm of captured by the orthogonal features.

collapse into trivial solutions if they are different in any aspects of the data items to provide high-quality representations for By doing so, the learned features can capture different aspects solution. To achieve the same effect, it aims at learning data sampling to avoid the potential issue of collapsing into a trivial for self-supervised representation learning.

C. Redundancy regularization

Redundancy regularization is a recently proposed technique for self-supervised representation learning. Unlike contrastive learning, redundancy regularization does not rely on negative sampling to avoid the potential issue of collapsing into a trivial solution. To achieve the same effect, it aims at learning data representations where features are orthogonal to each other. By doing so, the learned features can capture different aspects of the data items to provide high-quality representations for measuring similarity. Representations are also unlikely to collapse into trivial solutions if they are different in any aspects captured by the orthogonal features.

Barlow Twins (BT) is the representative method that leverages redundancy regularization. The pre-training algorithm of BT has an identical structure with the SimCLR algorithm as Figure 6 illustrates. Similar to SimCLR, the BT algorithm first applies DA operators to generate two different views $X_{ori}$ and $X_{aug}$ of the same batch $X$ of data items. BT applies the orthogonal to each other thus the off-diagonal values $C_{ij}$ for $i \neq j$ should be close to 0. To sum up, to learn effective representations, the cross-correlation matrix should be close to the $d$-dimensional identity matrix $I$. The loss function of BT is then defined as:

$$L_{BT} := \sum_{i=1}^{d} (1 - C_{ii})^2 + \lambda \sum_{i=1}^{d} \sum_{j \neq i} C_{ij}^2$$  (5)

where $\lambda$ is a hyper-parameter balancing the weights of terms.

To integrate BT into the pre-training process, we can combine the contrastive loss (Equation 2) with $L_{BT}$ linearly. Formally, the combined loss function is

$$L_{Sudowoodo} := (1 - \alpha) L_{Contrast} + \alpha L_{BT}$$  (6)

where $\alpha$ is the hyper-parameter controlling the weights of two loss functions. For the pre-training algorithm, we only need to replace Line 9 of Algorithm 1 with Equation 6 to compute the combined loss $L_{Sudowoodo}$.

To address this challenge, Sudowoodo generates more effective negative examples by sampling pairs of data items that are lexically similar. By doing so, we can train the representation model “harder” and force it to capture the semantic meaning of data items beyond lexicons. We use the standard TF-IDF (Term frequency - Inverse document frequency) method to obtain a sparse vector representation for each data item. We use the standard TF-IDF method to obtain a sparse vector representation for each data item. We then use the cosine similarity of the sparse vectors to measure the lexical similarity between each pair.

The process for generating similar pairs needs to be efficient since the input unlabeled corpus can be quite large and we need to repeatedly sample training batches for every training epoch. To this end, we choose the k-means algorithm which has running time linear to the dataset size $|D|$ and the number of clusters $k$. We add additional shuffling steps among and within the clusters to generate batches with more diversity. The negative sampling procedure is outlined in Algorithm 2 (replacing Line 5 of Algorithm 1).

**Algorithm 2:** Clustering-based negative sampling

**Input:** A dataset $D$ of serialized data items

**Variables:** Number of clusters $k$; Batch size $N$

**Output:** A set of mini-batches $\{B_1, B_2, \ldots, B_{|D|/N}\}$

1. $F \leftarrow$ TF-IDF-Featurize($D$);
2. /* Clustering and shuffling. Cache the results for future epochs */
3. Clusters $\leftarrow$ k-means($D, F, k$);
4. Batches $\leftarrow \emptyset$;
5. for $C \in$ Clusters do
6.   $C \leftarrow$ shuffle($C$);
7.   for $x \in C$ do
8.     $B_{last} \leftarrow$ $B_{last} \cup \{x\}$;
9.     if $|B_{last}| = N$ then
10.    Batches $\leftarrow$ Batches $\cup \{B_{last}\}$;
11.    $B_{last} \leftarrow \emptyset$;
12. return shuffle(Batches);

For the representations to be effective, we would like the same feature to be strongly correlated to itself in the $d$-dimensional representation space, the cross-correlation matrix is a $d$-by-$d$ matrix where each element is defined as

$$C_{ij} := \frac{\sum_{b=1}^{N} (z_{ori}^{b,i} z_{aug}^{b,j})^2}{\sum_{b=1}^{N} (z_{ori}^{b,i})^2 \cdot \sum_{b=1}^{N} (z_{aug}^{b,j})^2}$$  (4)

for dimension $i$ and $j$ in the representation space. Here an element $z_{ori}^{b,i}$ or $z_{aug}^{b,i}$ refers to the $i$-th dimension value of the $b$-th row in the batch $Z_{ori}$ and $Z_{aug}$ respectively. By this definition, the element $C_{ij}$ captures the cosine similarity between feature $i$ and $j$.

In order for the representations to be effective, we would like the same feature to be strongly correlated to itself in the two different views so the diagonal values $C_{ii}$ should be as close to 1 as possible. On the other hand, different features should be orthogonal to each other thus the off-diagonal values $C_{ij}$ for $i \neq j$ should be close to 0. To sum up, to learn effective representations, the cross-correlation matrix should be close to the $d$-dimensional identity matrix $I$. The loss function of BT is then defined as:

$$L_{BT} := \sum_{i=1}^{d} (1 - C_{ii})^2 + \lambda \sum_{i=1}^{d} \sum_{j \neq i} C_{ij}^2$$  (5)

where $\lambda$ is a hyper-parameter balancing the weights of terms.

To integrate BT into the pre-training process, we can combine the contrastive loss (Equation 2) with $L_{BT}$ linearly. Formally, the combined loss function is

$$L_{Sudowoodo} := (1 - \alpha) L_{Contrast} + \alpha L_{BT}$$  (6)

where $\alpha$ is the hyper-parameter controlling the weights of two loss functions. For the pre-training algorithm, we only need to replace Line 9 of Algorithm 1 with Equation 6 to compute the combined loss $L_{Sudowoodo}$.
V. Extension to More Tasks

We have introduced how to support different sub-tasks in Entity Matching with Sudowoodo in the previous sections. In this section, we generalize Sudowoodo to more DI&P tasks, namely data cleaning and column type detection.

A. Data cleaning

Data cleaning refers to the process of detecting and correcting corrupt, inconsistent, or missing data records from dirty data sources such as spreadsheets or relational tables. It is an important data preparation task for improving data quality for downstream applications [38], [39]. A data cleaning pipeline typically consists of two stages: error detection (ED) [11], [14] and error correction (EC) [13]. While many existing works focus on optimizing only one stage at a time, Sudowoodo provides a holistic solution for generating data corrections directly from the potentially contaminated data.

We consider the following problem setting. Given an input dirty relational table $T$ of $n$ rows with $m$ attributes, for each row $r \in T$ and each cell value $r_i \in r$, generate the correction $r_i^{c}$ for $r_i$ (if dirty) given $(r, T)$ as the context.

As Section II mentions, to apply Sudowoodo to the data cleaning problem, we first need a module that generates candidate corrections for each record. Sudowoodo follows the setting of Baran [13] which uses multiple external EC tools to generate the candidate corrections. These tools cover a wide variety of error types including missing value, typo, formatting issue, and violated attribute dependency (see Table III). Formally, for each cell value $r_i$, the EC tools generate a candidate correction set $\text{cand}(r_i)$. We note that Baran is a feature-based active learning method while Sudowoodo is a deep learning approach based on contrastive learning and language models.

To apply Sudowoodo in the data cleaning tasks, we first pre-train a representation model $M_{\text{emb}}$ with all cells and their candidate corrections as input. We consider two serialization schemes: contextual and context-free as in [36]. The context-free scheme serializes a cell $r_i$ as “[COL] attri [VAL] $r_i$”, where attri is the name of the $i$-th attribute. The contextual scheme serializes an entire row $r$ into

$[COL] \text{attr}_1 [VAL] r_1 \ldots [COL] \text{attr}_m [VAL] r_m$

and replaces $r_i$ with a corrected value in $\text{cand}(r_i)$ when encoding a candidate correction. Similar to EM, all optimizations (data augmentation, negative sampling, and redundancy regularization) for pre-training introduced in Section IV can be applied to this step.

A “blocking” phase serves as an optional step for further refining the candidate sets. This is usually not needed if the candidate sets are sufficiently small, but we note that there are cases where the refinement step is necessary. For example, the correction generator may choose to replace a misspelled token with words from a large vocabulary or may fill a missing value using a large domain (e.g., all possible city names).

Next, the domain experts label a few pairs of cell values with candidate corrections, i.e., $(r_i, r_i^c)$ where $r_i^c$ is a correction from the set $\text{cand}(r_i)$. The label is binary (1 or 0) indicating whether the correction is right or not. For data labeling, we do not apply the pseudo labeling step since the task is not similarity-based. Sudowoodo then fine-tunes the embedding model $M_{\text{emb}}$ into the pairwise matching model $M_{\text{pm}}$ using the same method introduced in Section III-B.

To finalize the correction results, for each cell $r_i$, we find the correction $r_i^c$ that maximizes the 1-probability as the model $M_{\text{pm}}$ predicts, i.e., $r_i^c = \arg \max_{r_i^c \in \text{cand}(r_i)} M_{\text{pm}}(r_i, r_i^c)$. The cell is considered clean if $r_i = r_i^c$ or $M_{\text{pm}}(r_i, r_i^c) = 0$. Otherwise, we output $r_i^c$ as the correction for $r_i$.

B. Semantic type detection

Given a table with several columns, the semantic type detection task considers assigning semantic types, such as “city”, “age”, and “population”, to each column [40], [41], [24], [42]. Existing learning-based methods consider the multi-class classification setting where the classifier assigns a single semantic type to each column [24], [15], and this formation is called as column type detection. It suffers from the challenge that domain experts need to conduct a challenging labeling task of choosing labels from a large candidate set. For example, in a recent semantic type detection method Sato [15], there are in total 78 possible types which can make the labeling task quite difficult.

In Sudowoodo, we take a different approach in the form of column matching to solve the semantic type detection task. Sudowoodo finds pairs of columns that have the same semantic type from a large collection of tables (e.g., the WebTable corpora [40]). These identified pairs can form clusters of same-type columns for which domain experts can easily assign the type by inspecting a few elements. This problem definition is closely related to the task of detecting pairs of related table columns [41] and domain discovery of table values [43]. Unlike existing methods [42], [24], Sudowoodo performs column matching thus it does not depend on pre-defined semantic types. Sudowoodo is also self-supervised requiring significantly less labeling effort.

We can adapt Sudowoodo to this task as follows. Here, each data item corresponds to a table column which can be serialized by concatenating the column values, for example:

[VAL] New York [VAL] California [VAL] Florida

similarly to EM where “[VAL]” indicates the start of a new column value. Note that to illustrate the effectiveness of Sudowoodo, we choose the bare-bone serialization method that does not take meta-information such as column names, adjacent columns, or table descriptions as input. The meta-information can be integrated into this process by modifying the serialization scheme, for example, by adding the column name to the left of the sequence.

We also need to adjust the DA operators before applying the pre-training algorithm of Sudowoodo here. The attribute-level operators no longer apply. In addition to the remaining token and span-level operators, we introduce a cell-level operator that shuffles the order of the column values. We can apply the rest of components in Sudowoodo without modification.
TABLE II: Statistics of EM datasets.

| Datasets       | TableA | TableB | Train+Valid | Test | %pos |
|----------------|--------|--------|-------------|------|------|
| Abt-Buy (AB)   | 1,081  | 1,092  | 7,659       | 1,916| 10.7%|
| Amazon-Google (AG) | 1,363  | 3,226  | 9,167       | 2,293| 10.2%|
| DBLP-ACM (DA)  | 2,616  | 2,294  | 9,890       | 2,473| 18.0%|
| DBLP-Scholar (DS) | 2,616  | 64,263 | 22,965      | 5,742| 18.6%|
| Walmart-Amazon (WA) | 2,554  | 22,074 | 8,193       | 2,049| 9.4% |

Finally, Sudowoodo outputs clusters of table columns that are inter-connected as the pairwise matching model \( M_{pm} \) predicts. By adjusting the column clustering algorithm, Sudowoodo can generate clusters of different granularity levels and identify semantic types that are not previously defined in the multi-class classification setting.

VI. EXPERIMENTS

We evaluate the performance of Sudowoodo on real-world datasets of entity matching (EM) and data cleaning. Specifically, for EM, we evaluate the performance of both the blocking and matching stages. For data cleaning, we combine the error detection and correction stages and evaluate the quality of the final corrections. We also conduct a case study of column semantic type discovery on the VizNet dataset. Due to space limitation, we provide more results on running time, parameter sensitivity, data profiling, and error analysis in the appendix of the technical report [44].

A. Experiment settings

For EM, we use 5 datasets provided by DeepMatcher [9] which are widely used in previous studies. These datasets are for training and evaluating EM models for various domains including products, publications, and businesses. Each dataset consists of two entity tables A and B to be matched. Blocking methods take these 2 tables as input and generate candidates of matched pairs. For matching, each dataset provides sets of labeled entity pairs where each pair has a binary label indicating whether it is a match / non-match. The goal is to decide whether each pair represents the same real-world object, e.g. publication or product. The original datasets are split into training, validation, and test sets at a 3:1:1 ratio. Since Sudowoodo targets application scenarios with insufficient label examples, we consider the settings of semi-supervised and unsupervised EM where Sudowoodo only uses 500 or 0 labels from the training and validation set. Table II shows the statistics of the datasets.

For data cleaning, we use 4 benchmark datasets provided by previous studies [13], [11]. Each dataset consists of a dirty spreadsheet and the goal is to generate a correction for each erroneous cell (Error Correction, EC). A related task is to identify cells that contain errors, called Error Detection (ED). Following the settings of Baran, Sudowoodo uses the same external EC modules for generating candidate corrections for each clean/dirty cell. Sudowoodo then utilizes the methods introduced in Section V-A to obtain the pairwise model for matching cells with their candidate corrections. The details of datasets are shown in Table III.

TABLE III: Statistics of data cleaning datasets. The error types are missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD) [13]. *Note that the tax dataset is sampled from the original dataset of 200k rows.

| Datasets       | size    | %error | Error Types     | Coverage | #candidates |
|----------------|---------|--------|-----------------|----------|-------------|
| beers          | 2,410 × 11 | 16%    | MV, FI, VAD     | 94.9%    | 63.4        |
| hospital       | 1,000 × 20 | 3%     | T, VAD          | 89.5%    | 68.3        |
| rayyan         | 1,000 × 11 | 9%     | MV, T, FI, VAD  | 51.4%    | 215.6       |
| tax*           | 5,000 × 15 | 4%     | T, FI, VAD      | 92.7%    | 1,442.3     |

1) Baseline methods: We compare Sudowoodo with SOTA EM methods DeepMatcher and Ditto and also 3 specialized methods for semi-supervised or unsupervised EM:

DeepMatcher [9] is a hybrid deep learning method based on Recurrent Neural Network (RNN) and the attention mechanism. We report the scores from the original paper.

Ditto [10] is the SOTA matching solution based on pre-trained LMs. Ditto has a series of optimizations including data augmentation (DA). We choose RoBERTa [45] as the base LM for Ditto which was shown to achieve the best performance.

Rotom [36] is a semi-supervised approach that leverages meta-learning to combine augmentations from multiple DA operators. Rotom was shown to achieve the SOTA results on EM in the low-resourced setting.

ZeroER [46] is an unsupervised EM solution based on Gaussian Mixture Models (GMM) for learning match/unmatch distributions.

Auto-FuzzyJoin [47] is an unsupervised approach that automatically builds fuzzy join programs. This method relies on the assumption that either Table A or B is a reference table with no or few duplicates.

For the blocking stage of EM, we compare Sudowoodo with DL-Block [22], a recently proposed deep learning framework achieving state-of-the-art (SOTA) results on blocking. DL-Block follows DeepMatcher and leverages a variety of deep learning techniques including self-supervised learning.

For data cleaning, we compare Sudowoodo with the EC method Baran [13]. Baran leverages active learning to learn an ensemble model of multiple external EC methods. The original setting of Baran assumes a perfect ED output or uses Raha [11] as the separate ED stage.

For column matching, we consider two SOTA baseline methods Sherlock [24] and Sato [15]. Both methods use ML/DL such as word2vec and LDA to represent columns as dense vectors.

For all the above baseline methods, we use the source code from the original code repositories to produce the results.

2) Environment and hyper-parameters: We implemented Sudowoodo using PyTorch [48] and Huggingface [49]. Unless otherwise specified, we use the RoBERTa-base model [45] as the pre-trained LM and AdamW as the optimizer for all the experiments. We fix the projector dimension to 768 and the pre-training related hyper-parameters \((\lambda, \tau)\) to (0.9, 3, 0.07). We pre-trained each model using 3 epochs and fine-tuned it for 50 epochs. We fix the size of the pre-training corpus to 10,000 by up or down-sampling the set of all data items. We set the
batch size to 64 and the learning rate to 5e-5. The matching and data cleaning tasks use the F1 score as the main evaluation metric. We list the hyper-parameters related to Sudowoodo’s optimizations in Table IV. For each run of the experiment, we select the epoch with the highest F1 on the validation set and report results on the test set. All experiments are run on a server machine with a configuration similar to a p4d.24xlarge AWS EC2 machine with 8 A100 GPUs.

**TABLE IV: Hyper-parameters of Sudowoodo and their choices.** We underlined the best combination found via grid search.

| Hyper-Parameter | Meaning | Range                        |
|-----------------|---------|------------------------------|
| cutoff_ratio    | % tokens to apply the cutoff DA | [0.01, 0.03, 0.05, 0.08]     |
| num_clusters    | Number of clusters for neg. sampling | [30, 60, 90, 120]            |
| alpha_pl        | Weight of the redundancy regularizer | (0.1, 0.01, 1e-3, 1e-4)     |
| multiplier      | Size of the training set w. pseudo labels | [2, 4, 6, 8, 10]             |

**B. Main results for Entity Matching**

For semi-supervised learning, we set the label budget to 500 for Sudowoodo. We use the same 500 labels for validation for further label saving. For the baselines Ditto and Rotom, we allow for 250 more training instances (in total 750) and compare the results with Sudowoodo with 500 labels to show the label efficiency of Sudowoodo. We fix the DA operator to be token del with span cutoff. For pseudo-labeling, we tuned the positive/negative thresholds ($\theta^+$, $\theta^-$) and found that adding 7x extra labels (i.e., multiplier = 8) works the best with the clustering optimization. We fix the number of fine-tuning steps unchanged when adding the extra labels.

We also conduct an ablation analysis to show the effectiveness of each optimization. We remove each of the following optimizations at a time to test their overall effectiveness:

- **PL**: the pseudo labeling technique (Section III-C);
- **Cls**: the clustering-based negative sampling (Section IV-B);
- **Cut**: the cutoff operator for DA (Section IV-A);
- **RR**: the redundancy regularization technique (Section IV-C).

We denote each variant by “Sudowoodo (-X, -Y, . . .)” indicating that optimizations X, Y, etc. are turned off. Note that SimCLR [23] is equivalent to Sudowoodo with all 4 optimizations turned off.

Table V summarizes the results under the setting of semi-supervised learning. When using 500 labels only, Sudowoodo achieves the overall best results and has a performance gain over Rotom (500) by up to 16%. While Rotom is already a label-efficient solution, Sudowoodo can achieve comparable F1 scores (78.3 vs. 78.5) by using 1/3 fewer labels (vs. Rotom (750)). This F1 score is also close to DeepMatcher (full) using 23x more labels.

Pseudo-labeling (PL) is the most effective optimization among the 4 tested options as removing it causes near 10% performance degradation. The 3 other optimizations, Cls, Cut and, RR, are especially effective on the more challenging WA dataset. All optimizations combined achieved up to 25% F1 improvement on the WA dataset or 11.2% on average.

**Unsupervised EM.** Table VI shows results of unsupervised EM. Note that for pseudo labeling, Sudowoodo requires prior knowledge of the positive label ratio which is available as a dataset statistic. Sudowoodo requires no more supervision beyond it. The results for ZeroER and Auto-FuzzyJoin are numbers either taken from the original papers or obtained by running the open-sourced code, whichever is higher.

Compared to the two SOTA methods, Sudowoodo outperforms ZeroER by up to 22.8% (or 7.7% on average) and Auto-FuzzyJoin by up to 34.8% (or 8.9% on average) F1 score respectively. The average performance of the unsupervised Sudowoodo is only 4% lower than that with 500 labels. Sudowoodo’s optimizations are again beneficial and they improve the base performance by 0.9% on average. The main source of the performance gain is again from the high-quality pseudo labels which are only slightly worse (<6%) than the semi-supervised setting. These results further illustrate the advantage of Sudowoodo in the low-resource scenarios.

**EM Blocking.** For blocking, Sudowoodo generates candidate pairs by applying kNN search over the learned vector representations of the right entity table (Table B) for k = 1 to 20. We find Sudowoodo achieves the best results using the DistilBERT [50] LM and a projector size of 4,096. We pre-train Sudowoodo for 5 epochs and use the last model checkpoint to generate candidates for evaluation.

We compare the quality of the candidate sets with the SOTA learning-based blocking method DL-Block. Following [22], we report the recall scores and the candidate set size ratio (CSSR) which equals the candidate set size divided by the total number of candidates (i.e., |TableA| × |TableB|). For each dataset, we compute the recall score as the fraction of positive pairs from the training, validation, and test sets that are retained by the candidate set.

Figure 7 shows the Recall-CSSR curves and Table VII shows the recalls and candidate set sizes of DL-Block and Sudowoodo. Here DL-Block’s numbers are taken from [22]. In Table VII, we report Sudowoodo’s (recall, #cand) for the
first $k$ where Sudowoodo’s recall is higher than DL-Block. The results show that Sudowoodo can consistently achieve the same recall level of DL-Block while having a significantly smaller candidate set (up to 84.8% smaller, AB). By having a smaller candidate set, Sudowoodo reduces the difficulty of the matching stage which can lead to further matching performance improvement or saving of human efforts. These results confirm Sudowoodo’s versatility in benefiting multiple sub-tasks of the EM application.

We provide ablation study, running time, data profiling and error analysis in Appendix A, B and E respectively in our technical report version [44].

TABLE VII: Sudowoodo for blocking. We report the recall score (R) and the number of candidates pairs (#cand).

|                | AB R #cand | AG R #cand | DA R #cand | DS R #cand | WA R #cand |
|----------------|------------|------------|------------|------------|------------|
| DL-Block       | 87.2       | 21,600     | 97.1       | 68,200     | 99.6       |
| Sudowoodo      | 88.6       | 2,376      | 97.3       | 48,390     | 99.6       |

C. Results for data cleaning

We evaluate Sudowoodo on data cleaning comparing it with the SOTA EC solution Baran [13]. We run Sudowoodo with all optimizations turned on except pseudo labeling. We use the span_shuffle DA operator with span cutoff for pre-training. We did not use blocking since the candidate correction sets are small (see Table III). After pre-training, we fine-tuned the representation model on 20 uniformly sampled rows (same amount of supervision for Baran using active learning). We evaluate the F1 score of Sudowoodo on the remaining rows following the same setting of Baran.

Table VIII summarizes the results. Baran has two settings: Raha+Baran uses Raha [11] as the ED model run before Baran. The second setting (Perfect ED + Baran) assumes an ED model that perfectly identifies all the dirty cells. We also report a method RoBERTa-base which fine-tunes the LM without Sudowoodo’s pre-training step.

Sudowoodo achieves the best overall results even outperforming Baran with perfect ED by 2.1% on average. Sudowoodo outperforms Raha+Baran by a large margin and in 3 out of 4 datasets by up to 44% F1 score (hospital). The results indicate that Sudowoodo’s strategy of formulating EC as a matching problem is beneficial. The contrastive pre-training is also important because without this step, RoBERTa LM has a performance degradation of up to 15.6% (hospital) F1 score. These results show the potential of integrating Sudowoodo into a data cleaning pipeline in real-world scenarios.

We provide detailed ablation results of Sudowoodo on EC in Appendix D of our technical report version [44].

TABLE VIII: Error correction (EC) F1 scores for data cleaning. Sudowoodo does not rely on a separate Error detection (ED) stage.

| Dataset          | Raha + Baran | Perfect ED + Baran | RoBERTa-base | Sudowoodo |
|------------------|--------------|--------------------|--------------|-----------|
| Beers hospital   | 74.59        | 94.03              | 89.43        | 95.74     |
| Rayyan tax       | 74.60        | 94.03              | 89.43        | 95.74     |
| Average          | 74.60        | 94.03              | 89.43        | 95.74     |

D. Results for semantic type detection

We conduct a case study where Sudowoodo is applied to the column matching task on the VizNet dataset. Following the setting in [15], we extracted more than 80k tables with 119k columns from the dataset annotated with 78 semantic types. We apply the column matching approach described in Section V-B. We use kNN blocking with $k = 20$ to extract pairs of nearest columns as candidate matches. After blocking, we use the ground truth labels from the original dataset to label 2k uniformly sampled pairs for training where we labeled a pair to be “match” if and only if they have the same original semantic type. We split this dataset into the training/validation/test sets according to the 2:1:1 ratio.

We compare the matching quality of Sudowoodo with those of the SOTA methods, Sherlock and Sato. Table X shows the detailed results. Here, we use Sherlock and Sato to extract features of the input pairs of columns for training pairwise matching models of Logistic Regression (LR), SVM, Random Forest (RF), and Gradient Boosting Tree (GBT). Among the 4 models, GBT achieves the best F1 on the validation set. The results show that Sudowoodo outperforms both Sherlock and Sato with GBT classifiers by 4.5% and 3.7% respectively.

Using the learned column embedding and matching models, Sudowoodo discovers over 5k column clusters (semantic types) from the table corpus. By inspecting the results, we find that Sudowoodo can discover more fine-grained types that are not present in the 78 ground truth types (Table IX). Note that Sudowoodo achieves this result by using only 1k column pair labels instead of more than 80k multi-class single-column labels required by Sato or Sherlock.

We provide more detailed experiments in the technical report version (see Appendix C) to compare different methods in our technical report version [44].

VII. RELATED WORK

Data cleaning. Deep learning has recently achieved great success in data cleaning and integration [51]. While traditional
TABLE IX: Column clusters discovered by Sudowoodo. The shown values are the first elements of 5 randomly selected columns having the same type. The headers (name, club, ...) are the majority ground truth types from [24], [15]. Sudowoodo discovers fine-grained types (e.g., central EU city) beyond the original set of 78 labels. We also provide our interpretations of such clusters/types (the term after the “→”).

| name            | club     | language | state → | result → US State | result → ball game result | name → company name | weight | city → central EU city | result → baseball in-game event |
|-----------------|----------|----------|---------|-------------------|--------------------------|----------------------|--------|------------------------|---------------------------------|
| Gerhart, Kyle   | SDSM     | Polski   | TX      | Win               | F. Rowe Price Associates, Inc., Lone Pine Capital LLC | 50 lbs or less 38kg |        |                        |                                  |
| Hossfeld, Nick  | GAKW     | Spanish  | LA      | 3-1 L            | Trigran Investments, Inc. | 40 lbs             |        |                        |                                  |
| Dege, Henry     | WSM      | English (built-in) | AZ   | W 9-0          | Ichlan Associates Corp. | up to 25 lbs |        |                        |                                  |
| Carlisle, Brendan | DCM      | Turkish  | NJ      | Win               | Apple Inc.               | 5 lbs               |        |                        |                                  |
| Tatlow, Jedidah | DCM      | English (built-in) | TX   | Win               | F. Rowe Price Associates, Inc., Lone Pine Capital LLC | 50 lbs or less 38kg |        |                        |                                  |

Semantic type detection. - Semantic type detection refers to the task of assigning semantics-rich types to table columns to enhance functionalities of data exploration systems such as Microsoft Power BI [71] and Google Data Studio [72]. Previous methods include both ontology-based [40], [73], [74] and learning-based methods [41], [75], [24], [15]. In particular, Sato [15] formulates semantic type detection as a multi-class classification problem with 78 pre-defined types and achieves the SOTA performance by leveraging conditional random field (CRF) and deep learning. Unlike Sato, Sudowoodo formulates the task as column matching and can detect fine-grained semantic types beyond the pre-defined set of classes.

Contrastive learning. - Contrastive learning has been a popular and effective technique in self-supervised representation learning for many applications. It is first applied in computer vision related tasks [76], [77], [78], [23], [26] based on the intuition that good representations should be invariant under different distortions. Contrastive learning has also been recently applied to NLP tasks by introducing a series of NLP-specific data augmentation operations. CERT [79] extends the idea of MoCo [76] to utilize back-translation for data augmentation. CLUTR [80] borrows the idea from SimCLR [23] to jointly train representations using a contrastive objective and masked language modeling. ConSERT [81] adopts contrastive learning to fine-tune BERT model in an unsupervised way for multiple downstream tasks. To the best of our knowledge, our work is the first one to employ contrastive learning in data integration and preparation applications.

VIII. CONCLUSION

We propose Sudowoodo, a multi-purpose data integration and preparation (DI&P) framework based on contrastive learning. It pre-trains high-quality representation models that can be fine-tuned to downstream DI&P tasks using few or even no labels. Unlike previous studies that optimize a single DI&P task at a time, we show that Sudowoodo’s representation models are applicable to a range of DI&P tasks including Entity Matching (EM), data cleaning, and data discovery. Our experiment results show that Sudowoodo achieves multiple SOTA results under different supervision levels for both the blocking and matching step for EM. The promising results also extend to the applications of data cleaning and column matching, which shows Sudowoodo’s versatility. For future work, we plan to study the integration of Sudowoodo with other closely related learning paradigms such as distant supervision, active learning, and domain adaptation.
[51] L. Dong and T. Rekatsinas, “Data integration and machine learning: A natural synergy,” PVLDB, vol. 11, no. 12, pp. 2094–2097, 2018.

[52] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi, “A cost-based model and effective heuristic for repairing constraints by value modification,” in SIGMOD. ACM, 2005, pp. 143–154.

[53] S. Kolahi and L. V. S. Lakshmanan, “On approximating optimum repairs for functional dependency violations,” in ICDT. vol. 361. ACM, 2009, pp. 53–62.

[54] Z. Abedjan, C. G. Akcora, M. Ouzzani, P. Papotti, and M. Stonebraker, “Temporal rules discovery for web data cleaning,” PVLDB, vol. 9, no. 4, pp. 336–347, 2015.

[55] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, “Holoclean: Holistic data repairs with probabilistic inference,” PVLDB, vol. 10, no. 11, pp. 1190–1201, 2017.

[56] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, “Activelclean: Interactive data cleaning for statistical modeling,” Proc. VLDB Endow., vol. 9, no. 12, pp. 948–959, 2016.

[57] S. Thirumuruganathan, N. Tang, M. Ouzzani, and A. Doan, “Data curation with deep learning,” in EDBT, 2020, pp. 277–286.

[58] L. Getoor and A. Machanavajjhala, “Entity resolution: Theory, practice & open challenges,” PVLDB, vol. 5, no. 12, pp. 2018–2019, 2012.

[59] P. Konda, S. Das, P. S. G. C., A. Doan, and et al., “Magellan: Toward building entity matching management systems,” PVLDB, vol. 9, no. 12, pp. 1197–1208, 2016.

[60] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu, “Corleone: hands-off crowdsourcing for entity matching,” in SIGMOD, 2014, pp. 601–612.

[61] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Baghavendh, and Y. Park, “Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services,” in SIGMOD, 2017, pp. 1431–1446.

[62] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi, “Schema-agnostic progressive entity resolution,” IEEE Trans. Knowl. Data Eng., vol. 31, no. 6, pp. 1208–1221, 2019.

[63] G. Simonini, S. Bergamaschi, and H. V. Jagadish, “BLAST: a loosely schema-aware meta-blocking approach for entity resolution,” PVLDB, vol. 9, no. 12, pp. 1173–1184, 2016.

[64] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, “Low-resource deep entity resolution with transfer and active learning,” in ACL, 2019, pp. 5851–5861.

[65] H. Nie, X. Han, B. He, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong, “Deep sequence-to-sequence entity matching for heterogeneous entity resolution,” in CIKM, 2019, pp. 629–638.

[66] C. Fu, X. Han, J. He, and L. Sun, “Hierarchical matching network for heterogeneous entity resolution,” in IJCAI, C. Bessiere, Ed., 2020, pp. 3665–3671.

[67] R. Peeters, C. Bizer, and G. Glavas, “Intermediate training of BERT for product matching,” in D2KG@VLDB, F. Piai, D. Firmani, V. Crescenzi, A. D. Angelis, X. L. Dong, M. Mazzei, P. Merialdo, and D. Srivastava, Eds., 2020.

[68] J. Wang, Y. Li, and W. Hirotou, “Machamp: A generalized entity matching benchmark,” in CIKM. ACM, 2021, pp. 4633–4642.

[69] J. Wang, Y. Li, W. Hirotou, and E. Kandogan, “Machop: an end-to-end generalized entity matching framework,” in aiDM ’22: Proceedings of the Fifth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. ACM, 2022, pp. 2:1–2:10.

[70] P. Wang, X. Zeng, L. Chen, F. Ye, Y. Mao, J. Zhu, and Y. Gao, “Promptem: Prompt-tuning for low-resource generalized entity matching,” Proc. VLDB Endow., vol. 16, no. 2, pp. 369–378, 2022.

[71] A. Ferrari and M. Russo, Introducing Microsoft Power BI. Microsoft Press, 2016.

[72] G. Snipes, “Google data studio,” Journal of Librarianship and Scholarly Communication, vol. 6, no. 1, 2018.

[73] O. Lehmborg, D. Ritzke, R. Meusel, and C. Bizer, “A large public corpus of web tables containing time and context metadata,” in WWW (Companion Volume). ACM, 2016, pp. 75–76.

[74] D. Ritzke and C. Bizer, “Matching web tables to dbpedia - A feature utility study,” in EDBT. OpenProceedings.org, 2017, pp. 210–221.

[75] K. Takeoka, M. Oyamada, S. Nakadai, and T. Okadome, “Meimeir: An efficient probabilistic approach for semantically annotating tables,” in AAAI. AAAI Press, 2019, pp. 281–288.

[76] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, “Momentum contrast for unsupervised visual representation learning,” in CVPR, 2020, pp. 9726–9735.

[77] X. Chen, H. Fan, R. B. Girshick, and K. He, “Improved baselines with momentum contrastive learning,” CoRR, vol. abs/2003.04297, 2020.

[78] T. Chen, S. Korobith, K. Swersky, M. Norouzi, and G. E. Hinton, “Big self-supervised models are strong semi-supervised learners,” in NeurIPS, 2020.

[79] H. Fang and P. Xie, “CERT: contrastive self-supervised learning for language understanding,” CoRR, vol. abs/2005.12766, 2020.

[80] J. M. Giorgi, O. Nitski, B. Wang, and G. D. Bader, “Dechnur: Deep contrastive learning for unsupervised textual representations,” in ACL/IJCNLP, 2021, pp. 879–895.

[81] Y. Yan, R. Li, S. Wang, F. Zhang, W. Wu, and W. Xu, “Consert: A contrastive framework for self-supervised sentence representation transfer,” in ACL/IJCNLP, 2021, pp. 5065–5075.