CollaborEM: A Self-Supervised Entity Matching Framework Using Multi-Features Collaboration

Congcong Ge, Pengfei Wang, Lu Chen, Xiaoze Liu, Baihua Zheng, and Yunjun Gao, *Member, IEEE*

Abstract—Entity Matching (EM) aims to identify whether two tuples refer to the same real-world entity and is well-known to be labor-intensive. It is a prerequisite to anomaly detection, as comparing the attribute values of two matched tuples from two different datasets provides one effective way to detect anomalies. Existing EM approaches, due to insufficient feature discovery or error-prone inherent characteristics, are not able to achieve stable performance. In this paper, we present CollaborEM, a self-supervised entity matching framework via multi-features collaboration. It is capable of (i) obtaining reliable EM results with zero human annotations and (ii) discovering adequate tuples' features in a fault-tolerant manner. CollaborEM consists of two phases, i.e., automatic label generation (ALG) and collaborative EM training (CEMT). In the first phase, ALG is proposed to generate a set of positive tuple pairs and a set of negative tuple pairs. ALG guarantees the high quality of the generated tuples, and hence ensures the training quality of the subsequent CEMT. In the second phase, CEMT is introduced to learn the matching signals by discovering graph features and sentence features of tuples collaboratively. Extensive experimental results over eight real-world EM benchmarks show that CollaborEM outperforms all the existing unsupervised EM approaches and is comparable or even superior to the state-of-the-art supervised EM methods.

Index Terms—Entity matching, sentence feature, graph feature, self-supervised, anomaly detection

1 Introduction

Due to widespread data quality issues [11], anomaly detection has received tremendous attention in diverse domains. It aims to find anomalous data in a dataset. Many studies [2], [53] focus on anomaly detection based on the information provided by a single dataset. Differently, we would like to highlight that the anomaly detection problem can be facilitated by considering the information captured by different sources, since it is common that different data sources can provide information about the same real-world *entity* [7]. Given two tuples from different relational datasets that refer to the same entity in real life, if they contain the same attribute but have contradictory values in the cell of the attribute, at least one of the values is an anomaly.

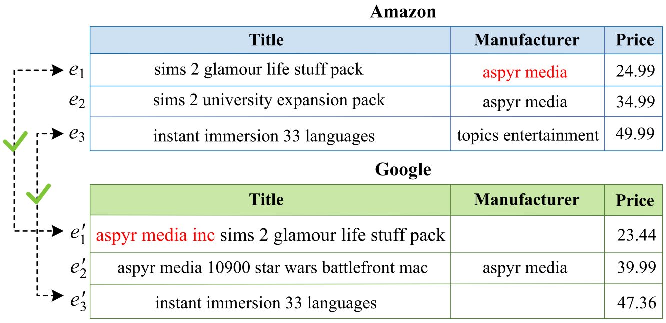
Example 1. Fig. 1 depicts two sampled tables, each of which contains three tuples about products gathered from Amazon and Google, respectively. In this figure, we assume the matched tuples (connected by tick-marked lines) that refer to the same real-world entity have been perfectly identified. Since e_1' and e_1 are matched tuples, it is expected that the two tuples share the same values for a given attribute. However, after comparing the attribute values of e_1 with the attribute values of e_1', we can easily spot an anomalous value w.r.t. the attribute *Title* of tuple e_1', which might be caused by data extraction errors. Specifically, the value "aspyr media inc" of e_1', corresponding to the attribute *Manufacturer*, is wrongly extracted into the cell of attribute *Title*.

Considering that it is impractical to assume all the matched tuples are known beforehand, in this paper, we focus on Entity Matching (EM) [7], which aims to identify whether a tuple from one relational dataset and another tuple from a different relational dataset refer to the same real-world entity. It is worth noting that reliable EM results are a prerequisite to ensure the quality of multi-source-based anomaly detection. This is because anomaly detection on top of false-aligned tuples is meaningless.

Early EM approaches require rules [12], [37] or crowdsourcing [14], [30], [44], which are impractical for matching real-world entities with literal or symbolic heterogeneity. Recently, embedding has become an increasingly powerful tool to encode heterogeneous entities into a unified semantic vector space, giving birth to various embedding-based EM techniques. Current embedding-based solutions to EM mostly rely on either *sentence features* or *graph features*. The former [10], [13], [18], [23], [26], [33], [52] treats each tuple as a sentence and learns the tuple's embedding according to the contextual information contained in the sentence. The latter [5], [24] first constructs graphs to represent tuples and then learns matching signals of tuples based on the graph structure. Despite the considerable EM performance on several benchmarks, identifying tuples referring to the same

- *Congcong Ge, Lu Chen, Xiaoze Liu, and Yunjun Gao are with the College of Computer Science, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: {gcc, luchen, xiaoze, gaoyj}@zju.edu.cn.*
- *Pengfei Wang is with the School of Software, Zhejiang University, Hangzhou, Zhejiang 310027, China. E-mail: wangpf@zju.edu.cn.*
- *Baihua Zheng is with the School of Computing and Information Systems, Singapore Management University, Singapore, Singapore 178902, Singapore. E-mail: bhzheng@smu.edu.sg.*

Manuscript received 29 Apr. 2021; revised 19 Nov. 2021; accepted 30 Nov. 2021. Date of publication 13 Dec. 2021; date of current version 8 Nov. 2023.
This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFC3300303, NSFC under Grants 62025206, 61972338, and 62102351, and also supported in part by the AZFT.
(Corresponding author: Yunjun Gao.)
Recommended for acceptance by J. Li, L. He, H. Peng, P. Cui, C.C. Aggarwal, and P.S. Yu.
Digital Object Identifier no. 10.1109/TKDE.2021.3134806



Amazon

	Title	Manufacturer	Price
e_1	sims 2 glamour life stuff pack	aspyr media	24.99
e_2	sims 2 university expansion pack	aspyr media	34.99
e_3	instant immersion 33 languages	topics entertainment	49.99

Google

	Title	Manufacturer	Price
e'_1	aspyr media inc sims 2 glamour life stuff pack		23.44
e'_2	aspyr media 10900 star wars battlefront mac	aspyr media	39.99
e'_3	instant immersion 33 languages		47.36

Fig. 1. Example of using EM for anomaly detection.

real-world entity, however, is still a challenging endeavor. The challenges are mainly two-fold, as listed below.

Challenge I: *Labor-intensive annotations for generating prematched tuples*. Embedding-based EM can achieve considerable results but typically requires a large number of labeled tuple pairs. The annotating process is labor-intensive and hence restricts the scope of its applications in real-world EM scenarios. Although several EM approaches [5], [46], [50] have tried to perform EM in an unsupervised way without any annotation, their EM performance is far from satisfactory due to the error-sensitive nature. A large body of research [3], [15], [27] has indicated that unsupervised methods can be easily misled/fooled or attacked since they do not include any supervision signal. Therefore, even slight erroneous data may lead to wrong results. For example, ZeroER [46], the state-of-the-art unsupervised EM method, achieves poor performance on dirty datasets, as confirmed in the experiments to be presented in Section 6.3. Since real-world EM datasets usually incorporate various errors, it is challenging to apply unsupervised methods to solve real-world EM tasks directly.

Challenge II: *Insufficient feature discovery of the tuples for EM*. Based on our preliminary study, neither sentence-based nor graph-based approaches can discover sufficient features of tuples to achieve high-quality EM results. To ease the understanding of these two types of approaches, we detail their respective strengths and limitations in the following.

For the sentence-based methods, the embedding of a tuple is highly relevant to its serialized attribute values. It is resilient to anomalous values caused by data extraction errors. Take the tuple e'_1 in Fig. 1 as an example. The attribute value of *manufacturer* (i.e., “aspyr media inc”) appears in a different place (as a part of attribute *title* instead of *manufacturer*), due to data extraction errors. The sentence-based methods treat the tuple e'_1 as a sentence *“aspyr, media, inc, sims, 2, glamour, life, stuff, pack, 23.44”*. In other words, “aspyr media inc” is still a part of the context of e'_1 and can provide effective information to learn the embedding of e'_1. Despite the benefit, two main limitations exist in the sentence-based methods. First, recent work [5] clarifies that, tuples are not sentences, and hence, treating a tuple blindly as a sentence loses a large amount of contextual information present in the tuple. Second, they dismiss the rich set of semantics inherent among different tuples [5]. To be more specific, they assume that different tuples are mutually independent. On the contrary, it is common that different tuples share the same attribute values, and some common attribute values might appear in many tuples. As shown in Fig. 1, the attribute value “aspyr media” exists in both tuple e_1 and tuple e_2.

On the other hand, the graph-based EM approaches bring two benefits. First, it can capture the semantic relationships between different attributes within every tuple. Second, it can discover the rich set of semantics inherent among different tuples. Recent studies [5], [24] transform every dataset containing a collection of tuples into a graph composed of three types of nodes, i.e., *tuple-level nodes*, *attribute-level nodes*, and *value-level nodes*. The graph exhibits two characteristics: (i) there is an edge between a tuple-level node and a value-level node as long as the value appears in the tuple; and (ii) there is an edge between an attribute-level node and a value-level node if the value belongs to the domain of this attribute. However, graph-based EM is error-prone. Take the sampled Amazon dataset in Fig. 1 as an example. The value “aspyr media inc”, which corresponds to a wrong attribute-level node, will result in a wrong graph structure. The wrong graph features might be propagated along the edges and nodes, and thus lead to unreliable embeddings of tuples. Consequently, we are required to find sufficient tuple features in order to equip graph-based EM approaches with fault-tolerance.

Contributions. The obstruction with the existing EM methods inspires us to ask a question: would it be possible to perform EM in a *self-supervised* manner, where reliable labels are automatically generated and sufficient entities features are captured, so that the above two challenges could be well addressed? Accordingly, we propose CollaborEM, a self-supervised entity matching framework powered by multi-features collaboration. CollaborEM features a sequential modular architecture consisting of two phases, i.e., *automatic label generation (ALG)* and *collaborative EM training (CEMT)*. In the first phase, ALG is developed to generate reliable EM labels on every dataset automatically. In the second phase, with the guidance of the generated labels, CEMT learns the matching signals by utilizing both *sentence features* and *graph features* of tuples collaboratively.

We summarize the contributions of this paper as follows:

- *Self-supervised EM framework*. We propose a self-supervised EM framework CollaborEM, which requires *zero* human involvement to generate labeled tuple pairs with high quality. Once the reliable labels are generated, CollaborEM produces the state-of-the-art EM results via the collaboration of both *sentence features* and *graph features* of tuples.
- *Automatic label generation*. We present ALG, for the first time, to automatically generate both *positive* and *negative* tuple pairs for the EM task. Also, ALG greatly helps CollaborEM to correctly identify “challenging” tuple pairs that are difficult to tell whether they are matched.
- *Collaborative EM training*. We present CEMT, a collaborative EM training approach, to discover both *graph features* and *sentence features* to learn sufficient tuple features for EM without sacrificing the fault-tolerance capability in handling noisy tuples.
- *Extensive experiments*. Comprehensive evaluation over eight existing EM benchmarks demonstrates

TABLE 1
Symbols and Description

Notation	Description
T	a relational dataset
$e \in T$	a tuple belonging to the dataset T
A	a set of attribute values
$e.A[m]$	the mth attribute value of tuple e
$\mathcal{G}$	a multi-relational graph
N	a set of nodes belonging to the graph $\mathcal{G}$
E	a set of edges belonging to the graph $\mathcal{G}$

the superiority of CollaborEM. It outperforms all the existing unsupervised methods. Furthermore, it is comparable with or even superior to the state-of-the-art supervised EM method.

Organization. The rest of the paper is organized as follows. Section 2 covers the basic background techniques used in the paper. Section 3 presents the overall architecture of our proposed CollaborEM, and Sections 4 and 5 detail the two key phases of CollaborEM respectively. Section 6 reports the experimental results and our findings. Section 7 reviews the related work. Finally, Section 8 concludes the paper.

2 Preliminaries

In this section, we first formalize the problem of entity matching and then overview some background materials and techniques to be used in subsequent sections. Table 1 summarizes the symbols that are frequently used throughout this paper.

2.1 Problem Definition

Let T be a relational dataset with $|T|$ tuples and m attributes $A = \{A[1], A[2], \ldots, A[m]\}$. Each tuple $e \in T$ consists of a set of attribute values, denoted as $V = \{e.A[1], e.A[2], \ldots, e.A[m]\}$. Here, $e.A[m]$ is the mth attribute value of tuple e, corresponding to attribute $A[m] \in A$. Entity Matching (EM), also known as entity resolution and record linkage, aims to identify whether two tuples from different datasets refer to the same real-world entity.

Generally, EM often executes a *blocking* phase followed by a *matching* phase [46]. Blocking is to reduce the quadratic number of candidates of matched tuple pairs. In other words, it produces a small subset of $T \times T'$ for candidate pairs with high probabilities to be matched. After blocking, matching aims to identify the true matched/unmatched tuple pairs in the candidate pairs. In this paper, we focus on the matching phase. Given two datasets T and T', CollaborEM is to assign a binary label $y \in \{0, 1\}$ for each tuple pair $(e, e') \in T \times T'$ with *zero* pre-defined labeled tuple pair. Here, $y = 1$ denotes a truly matched pair, and $y = 0$ represents a unmatched pair.

2.2 Pre-Trained Language Models

Pre-trained language models (LMs), such as BERT [8] and XLNet [48], have demonstrated a powerful semantic expression ability. Based on pre-trained LMs, we can support many downstream tasks (e.g., classification and question answering). Concretely, we can plug appropriate inputs and outputs into a pre-trained LM based on the specific task and then fine-tune all the model's parameters end-to-end.

Intuitively, the EM problem can be treated as a sentence pair classification task [26]. Given two tuples $e_i \in T$ and $e'_j \in T'$, pre-trained LMs transform them into two sentences $\mathcal{S}(e_i)$ and $\mathcal{S}(e'_j)$, respectively. A sentence $\mathcal{S}(e_i)$ is denoted by $\mathcal{S}(e_i) ::= \langle$[COL] $A[1]$ [VAL] $e_i.A[1]$ ... [COL] $A[m]$ [VAL] $e_i.A[m]\rangle$, where [COL] and [VAL] are special tokens for indicating the start of attribute names and the start of attribute values, respectively. Note that, we exclude missing values and their corresponding attribute names from the sentence since they contain zero valid information. A tuple pair (e_i, e'_j) can be serialized as a pairwise sentence $\mathcal{S}(e_i, e'_j) :: = \langle \mathcal{S}(e_i)$ [SEP] $\mathcal{S}(e'_j)\rangle$, where [SEP] is a special token separating the two sentences. For the classification task, pre-trained LMs take each pairwise sentence $\mathcal{S}(e_i, e'_j)$ as an input. Note that, a special symbol [CLS] is added in front of the input sentence. It is utilized to store the classification output signals during the fine-tuning of LMs.

Objective Function. We employ *CrossEntropy Loss*, a widely used classification objective function, to fine-tune the pre-trained LMs in CollaborEM. CrossEntropy Loss is designed to keep the predicted class labels similar to the ground-truth. Formally,

$$\mathcal{L}(y = k|\mathcal{S}(e_i, e'_j)) = -\log\left(\frac{\exp(d_k)}{\sum_q^{|k|}\exp(d_q)}\right)\forall k \in \{0, 1\}. \tag{1}$$

Here, $\boldsymbol{d} \in \mathbb{R}^{|k|}$ is the logits computed by $\boldsymbol{d} = \mathbf{W}_c^\top \mathbf{E}_{[\mathbf{CLS}]}$. $\mathbf{W}_c \in \mathbb{R}^{n\times|k|}$ is a learnable linear matrix, where n is the dimension of the sentence embeddings. $\mathbf{E}_{[\mathbf{CLS}]}$ is the embedding of the symbol [CLS]. As mentioned in Section 2.1, the class labels are binary {0, 1} for sentence pair classification. We denote $y = 1$ a truly matched tuple pair and $y = 0$ a unmatched tuple pair.

2.3 Graph Neural Networks

Graph neural networks (GNNs) are popular graph-based models, which capture graph features via message passing between the nodes of graphs. GNNs are suitable for the EM task because of the following two aspects. First, GNNs ignore the sequence relationship between different attributes but discover the features of each tuple by aggregating the semantic information contained in the corresponding attribute names and values. It conforms to the real characteristics of relational datasets. This is because tuples' attributes can be organized in any order rather than a specifically ordered sentence. Thus, GNNs are able to effectively capture the features within every tuple. Second, recall that GNNs capture graph features via message passing between relevant nodes, i.e., the set of tuples sharing the same attribute values in this paper. Accordingly, GNNs have the capability of learning rich semantics among those relevant tuples, since the features of a tuple can be passed to another tuple through an edge (i.e., a shared attribute value). The core idea of GNNs is to learn each node representation by capturing the information passing from its neighborhoods. Generally, GNNs learn the embeddings of each node n_i obeying the following equations [29]:

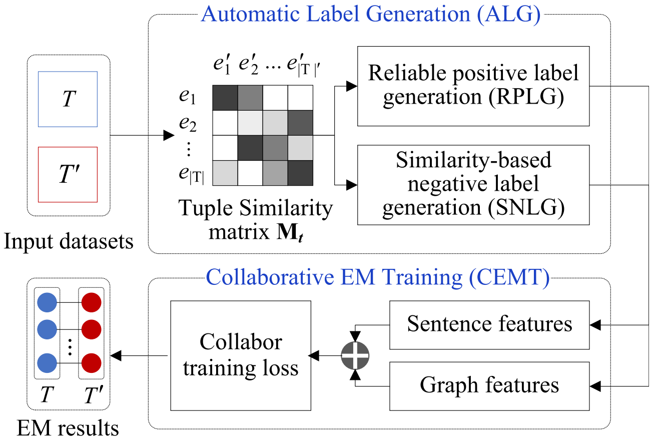

Automatic Label Generation (ALG)
T
T′
Input datasets
e'_1 e'_2 … $e'_{|T'|}$
e_1
e_2
⋮
$e_{|T|}$
Tuple Similarity matrix $\mathbf{M}_t$
Reliable positive label generation (RPLG)
Similarity-based negative label generation (SNLG)
Collaborative EM Training (CEMT)
Sentence features
Graph features
Collabor training loss
T
T′
EM results

Fig. 2. CollaborEM framework.

$$\mathbf{o}_i^{l+1} = \mathrm{AGGREGATION}^l\left(\left\{\!\!\left\{\left(\mathbf{h}_j^l, \mathbf{r}_{i,j}\right) : j \in \mathcal{N}(i)\right\}\!\!\right\}\right) \tag{2}$$

$$\mathbf{h}_i^{l+1} = \mathrm{UPDATE}^{l+1}\left(\mathbf{h}_i^l, \mathbf{o}_i^{l+1}\right) \tag{3}$$

where $\mathbf{h}_i^l$ represents the embedding of the lth layer node n_i, $\mathbf{r}_{i,j}$ stands for the embedding of an edge that connects the node n_i and another node n_j, and $\{\!\{\cdots\}\!\}$ denotes a multiset. $\mathcal{N}(i)$ represents the set of neighboring nodes around e_i. Eq. (2) is to aggregate information from the neighboring nodes while Eq. (3) transforms the entity embeddings into better ones. To serve the purpose of AGGREGATION, we can use graph convolutional network (GCN) [19] or graph attention network (GAT) [43].

3 Framework Overview

In this section, we overview the framework of CollaborEM, as illustrated in Fig. 2. CollaborEM consists of two phases, i.e., (i) automatic label generation (ALG) and (ii) collaborative EM training (CEMT).

Automatic Label Generation (ALG). As mentioned in Section 1, pre-collected EM labels are often not available in many real-world scenarios. It inspires us to look for ways to generate approximate labels via an automatic label generation program. Given two datasets T and T', each of which contains a collection of tuples, this phase is to generate pseudo-labels with high-quality, including both *positive labels* and *negative labels*, for the guidance of the subsequent CEMT process.

Positive labels refer to a set of positive tuple pairs, denoted as $\mathbb{P}$. For each positive tuple pair $\mathbb{P}(e_i, e'_i)$, the tuple $e_i \in T$ and the tuple $e'_i \in T'$ have a high probability of being matched. In ALG, we introduce a *reliable positive label generation (RPLG)* strategy to obtain positive tuple pairs with high confidence.

On the other hand, negative labels refer to a set of negative tuple pairs, denoted as $\mathbb{N}$. For each negative tuple pair $\mathbb{N}(e_i, e'_j)$, the tuple $e_i \in T$ and the tuple $e'_j \in T' - \{e'_i\}$ are unlikely to be matched. Random sampling [32], [42] is a widely-used approach for generating negative labels. Given a positive tuple pair $\mathbb{P}(e_i, e'_i)$, random sampling replaces either e_i or e'_i with an arbitrary tuple. However, recent studies [39], [40] have indicated that the randomly generated negative tuple pairs are easily distinguished from positive ones. For instance, if we generate a negative tuple pair ("Apple Inc.", "Google") for a positive tuple pair ("Apple Inc.", "Apple"), it is obvious that "Google" and "Apple Inc." are not equivalent. These negative tuple pairs are uninformative, and contribute little to the embedding training process. Ideally, an effective negative label generation is expected to put two similar tuples (but they are not related to the same real-world entity) into a pair. It facilitates an EM-oriented embedding model (e.g., CEMT in this paper) to be capable of identifying whether two entities of a "challenging" tuple pair refer to the same real-world entity. Thus, we propose a *similarity-based negative label generation (SNLG)* method in ALG to generate negative labels with semantic similarity.

Collaborative EM Training (CEMT). Recall that matching entities purely based on the *sentence features* or the *graph features* of tuples results in insufficient feature discovery or erroneous feature involvement. The goal of CEMT is to capture and integrate both the sentence features and the graph features of tuples in a unified framework to improve the quality of EM results. Given two datasets T and T' and a set of labels (including positive labels $\mathbb{P}$ and negative labels $\mathbb{N}$) generated by ALG, CEMT first introduces *multi-relational graph construction (MRGC)* to construct a multi-relational graph $\mathcal{G}$ (resp. $\mathcal{G}'$) for each dataset T (resp . T'). We would like to highlight that the graph structure generated by the proposed MRGC is much *simpler* than that generated by other existing EM methods (e.g., EMBDI [5] and GraphER [24]) without losing the expressive power of tuples' graph features, as confirmed in the experimental evaluations to be presented in Section 6.5.2.

Then, CEMT learns the embeddings of each tuple based on the graph structure. CEMT is treated as a black box, such that users could enjoy the flexibility of applying their choice of graph-based models to embed both nodes and edges in a multi-relational graph. Our current implementation utilizes AttrGNN [28] for this purpose. Afterward, we feed the well-trained graph features (i.e., embeddings) of tuples into a pre-trained LM to assist the learning of the sentence features of tuples. More specifically, the graph features of tuples are used to complement the semantic features of tuples that cannot be captured by a sentence-based model.

4 Automatic Label Generation (ALG)

In this section, we present an automatic label generation (ALG) strategy. It contains two components, including (i) a reliable positive label generation (RPLG) method and (ii) a similarity-based negative label generation (SNLG) method.

Generating either positive labels or negative labels is highly relevant to the similarity between tuples. Motivated by the powerful capability of semantics expression of pre-trained language models, we leverage sentence-BERT [35], a variant of BERT that achieves the state-of-the-art performance for semantic similarity search, to assign a pre-trained embedding for each tuple. In general, different similarity functions (e.g., cosine distance and Manhattan distance) can be applied to quantify the semantic similarity between tuples from different datasets in ALG, according to the characteristics of the datasets. In the current implementation, we find empirically that cosine distance brings considerable performance. To this end, we choose it as the similarity function in ALG. The tuple similarity matrix is denoted as $\mathbf{M}_t \in [0,1]^{|T|\times|T'|}$, where $|T|$ and $|T'|$ represent the total number of tuples in T and T' respectively. In the following, we detail how to generate positive and negative labels via RPLG and SNLG, respectively.

4.1 Reliable Positive Label Generation (RPLG)

RPLG aims to find positive tuple pairs with a high probability of being matched. A common approach is to consider tuples that are mutually most similar to each other. However, we find empirically many mutually similar tuples do not refer to the same entities. Considering that high-quality labels are essential for embedding-based EM model as wrong labels will mislead the EM model training, we generate the positive labels by IKGC [49], which gives much stronger constraints to ensure the high-quality of positive labels than the methods based on the mutual similarity. It generates tuple pairs as positive labels that satisfy two requirements [49]: (i) they are mutually most similar to each other; and (ii) there is a margin between, for each tuple e, its most similar tuple and the second most similar one.

Specifically, for each tuple $e_i \in T$, we assume that $e'_j, e'_k \in T'$ are the most similar and the second most similar tuples in T' to e_i, respectively. Similarly, for tuple $e'_j \in T'$ (i.e., the most similar tuple to $e_i \in T$), let e_l and e_u denote its most similar tuple in T and the second most similar tuple in T respectively. If tuple pair (e_i, e'_j) could be considered as a positive label, we expect $e_i = e_l$, i.e., e_i and e'_j are mutually most similar to each, i.e., the requirement (i) stated above. In addition, their similarity discrepancies $\delta_1 = Sim(e_i, e'_j) - Sim(e_i, e'_k)$ and $\delta_2 = Sim(e'_j, e_l) - Sim(e'_j, e_u)$ are expected to be both above a given threshold θ, i.e., the requirement (ii) stated above. Here, $Sim(e, e')$ denotes the similarity score between two tuples $e \in T$ and $e' \in T'$.

Discussion. We would like to emphasize that RPLG is a general approach, which can be easily integrated into various EM methods. RPLG is able to not only generate positive labels with high-quality (see the experiments to be presented in Section 6.5.1) but also achieve desirable EM results without any time-consuming training process (see the experiments to be presented in Section 6.4). In addition, to find similar objects of each tuple in RPLG, a prevalent solution is *nearest neighbor search* (NNS). Various approaches for NNS have been proposed, including exact methods (e.g., naive NNS) and approximation ones (e.g., locality sensitive hashing). In the current implementation, we apply the naive NNS for RPLG. Given a tuple $e \in T$, NNS computes exactly the similarity from e_i to every tuple $e' \in T'$. The time complexity of the naive NNS is $O(d \times |T'|)$, where $|T'|$ is the cardinality of a relational dataset T', and d is the dimensionality of tuple's embedding. Furthermore, since every tuple $e \in T$ needs to find its similar ones by applying the native NNS in RPLG, we can obtain that the time complexity of RPLG is $O(d \times |T| \times |T'|)$. The reason why we use exact method instead of other approximate ones is as follows. We found empirically that performing RPLG with the naive NNS only spends a few seconds on average over the experimental datasets. Compared with the entire training time of CollaborEM (w.r.t. the efficiency evaluation to be presented in Section 6.3), the running time of RPLG is almost negligible. However, approximation methods need a trade-off between the computational efficiency and the quality of solutions obtained. To this end, approximation methods may incur relatively inaccurate similarity results and hence conduct unreliable positive labels. Nonetheless, it is required to apply approximation NNS methods for dealing with large-scale datasets since exact methods are time-consuming. We left the problem of generating reliable positive labels with large-scale datasets as a future direction.

4.2 Similarity-Based Negative Label Generation (SNLG)

As the random-based negative label generation method has rather limited contribution to the embedding-based EM training, it is essential to generate more "challenging" negative labels, as described in Section 3. To achieve this goal, we propose a similarity-based negative label generation (SNLG) strategy. Given a positive tuple pair $\mathbb{P}(e_i, e'_i)$, where $e_i \in T$ and $e'_i \in T'$, SNLG generates a set of negative labels $\mathbb{N}(e_i, e'_i)$ by replacing either e_i or e'_i with its ϵ-nearest neighborhood in the semantic embedding space. Again, we use the cosine similarity metric to search for the ϵ-nearest neighbors of e_i and e'_i, respectively.

Discussion. Even though this is a very intuitive and simple method, it effectively promotes the performance of CollaborEM. We will demonstrate the superiority of using the proposed SNLG to generate negative labels for EM in the experiments to be presented in Section 6.4. In addition, considering that we have already computed the similarity between every tuple pair in RPLG, we can obtain the ϵ-nearest neighborhood of each tuple $e \in \mathbb{P}$ directly. To this end, the time complexity of SNLG is $O(|\mathbb{P}|)$, where $|\mathbb{P}|$ denotes the number of the positive labels.

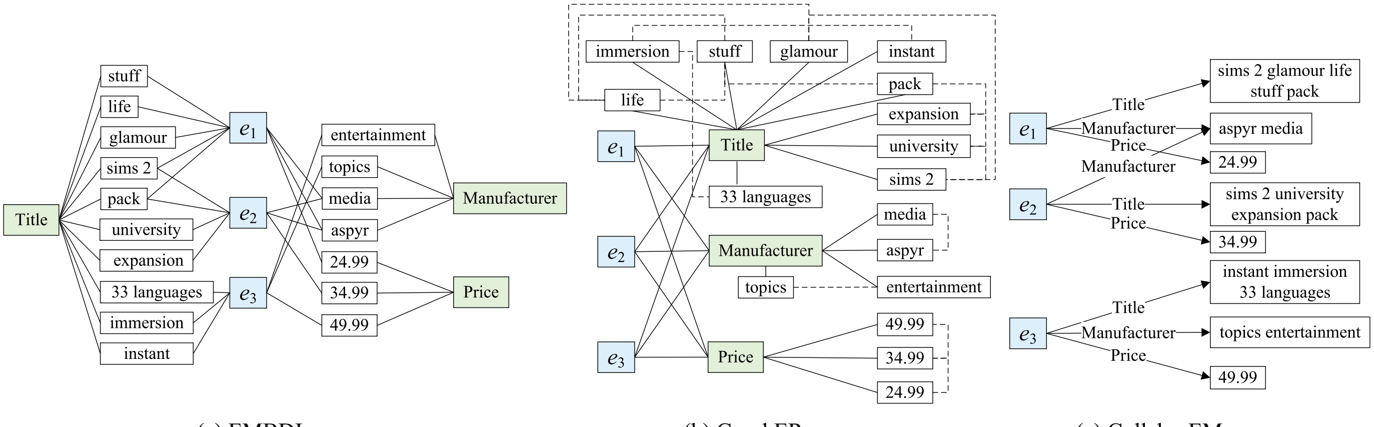
5 Collaborative EM Training (CEMT)

This section details a newly proposed collaborative EM training (CEMT) approach to discover the features of tuples from both the graph aspect and the sentence aspect to facilitate the EM process. CEMT is composed of two phases, i.e., (i) multi-relational graph feature learning (MRGFL) and (ii) collaborative sentence feature learning (CSFL).

5.1 Multi-Relational Graph Feature Learning (MRGFL)

Inspired by the graph structure's powerful capturing ability of semantics, we propose a multi-relational graph feature learning method (MRGFL) to represent tuples according to their graph features. It first proposes a *multi-relational graph construction* (MRGC) approach for transforming datasets from the relational format to the graph structure, and it then learns the tuple features via a GNN-based model, e.g., AttrGNN [28] in our current implementation.

Multi-Relational Graph Construction (MRGC). Graph construction techniques have been presented in the existing EM work, such as EMBDI [5] and GraphER [24]. These techniques treat tuples, attribute values, and attribute names as three different types of nodes. Edges exist if there are relationships between nodes. Nonetheless, several drawbacks restrict the scope of using these graph construction methods to perform EM in real-world scenarios.



(a) EMBDI (b) GraphER (c) CollaborEM

Fig. 3. A motivating example of proposing the multi-relational graph construction (MRGC).

First, these graph construction approaches produce intricately large-scale graphs containing a large number of edges and nodes. Storing a graph with massive edges and nodes is memory-consuming, and training a graph embedding model (e.g., GNN) on a large graph is challenging too, as widely-acknowledged by many recent studies [6], [16].

Second, these graph construction methods lack consideration of the semantics contained in an edge itself. For instance, assume that there are two types of edges, i.e., *attribute-value edges* and *tuple-attribute edges*. The former represents an edge that connects an attribute-level node with a value-level node; the latter represents an edge connecting a tuple-level node with an attribute-level node. It is intuitive that these two types of edges have different semantic meanings, and hence, they should be considered differently when learning features of tuples in the EM task.

Algorithm 1. Multi-Relational Graph Construction

```
  Input: a relational dataset $T$
  Output: a multi-relational graph $\mathcal{G}$
1 $\mathcal{G} \longleftarrow \varnothing$;
2 foreach $e_i \in T$ do
3   $\mathcal{G}$.addNode($e_i$);
4   foreach $v_j \in \{e_i.A[1], e_i.A[2], \ldots, e_i.A[m]\}$ do
5     if $v_j$ is not included in $\mathcal{G}$ then
6       $\mathcal{G}$.addNode($v_j$);
7     $a_{i,j} \longleftarrow$ find the attribute name of $v_j$;
8     $\mathcal{G}$.addEdge($e_i, a_{i,j}, v_j$);
9 return $\mathcal{G}$
```

The above limitations motivate us to design a relatively *small-scale* but highly *effective* MRGC to construct a multi-relational graph for every dataset. We start by defining a multi-relational graph, formally $\mathcal{G} = \{N, E, A\}$. Here, N and E refer to a set of nodes and a set of edges, respectively; and A represents the set of attributes corresponding to the nodes and the edges. There are two types of nodes in $\mathcal{G}$, i.e., *tuple-level nodes* and *value-level nodes*. A tuple-level node represents a tuple e; while a value-level node corresponds to an attribute value v in a relational dataset. Each attribute $a \in A$ denotes an attribute name in the relational dataset. $E = \{(e, a, v) | e, v \in N, a \in A\}$ represents the set of edges. Each edge connects a tuple-level node e with a value-level node v via an attribute a, meaning that e has v as its value for attribute a.

Next, we describe the MRGC procedure, with its pseudo code presented in Algorithm 1. Given a relational dataset T, MRGC initializes an empty multi-relational graph $\mathcal{G}$ (Line 1). Then, MRGC iteratively adds nodes and edges to $\mathcal{G}$ (Lines 2-8). For every tuple $e_i \in T$, MRGC first selects its tuple Id as a tuple-level node (Line 3) and then adds a set of value-level nodes that correspond to this tuple (Lines 4-6). Note that, since different tuples share the same attribute names, MRGC generates a set of edges for e_i, with each connecting the tuple-level node of e_i and a value-level node $v_j \in \{e_i.A[1], e_i.A[2], \ldots, e_i.A[m]\}$, denoted as $(e_i, a_{i,j}, v_j)$.

Discussion. Compared to the existing graph construction methods, MRGC constructs a small graph that is still able to well preserve the semantics of tuples. Take the sampled Amazon dataset as an example. Fig. 3 shows the respective graph structures constructed by three different graph construction methods, including EMBDI [5], GraphER [24], and the proposed MRGC in this paper. It is obvious that the graph constructed by MRGC is the smallest, containing fewer nodes and edges than other graphs. Also, we will verify the small-scale characteristics of MRGC in the experiments to be presented in Section 6.5.2. Besides, MRGC not only preserves the semantic relationships between each tuple and its corresponding attribute values, but also maintains semantic connections between different tuples by connecting them with a shared value-level node. For example, e_1 and e_2 have semantic connections since they both have edges linking to the same value-level node "aspyr media". The time complexity of MRGC is $O(|T| \times |A| + |T'| \times |A'|)$. Here, $|T|$ and $|T'|$ are the number of tuples in the relational datasets T and T', respectively; and $|A|$ and $|A'|$ are the cardinalities of the attributes $A \in T$ and $A' \in T'$, respectively.

Tuple Feature Learning. Given two multi-relational graphs $\mathcal{G}$ (w.r.t. T) and $\mathcal{G}'$ (w.r.t. T'), MRGFL aims to embed tuples from different sources in a unified vector space by considering their graph structures. In this vector space, matched tuples are expected to be as close to each other as possible. Generally, we can treat MRGFL as a graph-based EM problem, which is highly relevant to *entity alignment* (EA) [41] that aims to find a correspondence between entities from different multi-relational graphs. To this end, MRGFL is regarded as a black box. Users have the flexibility to learn the embeddings of tuples by applying any available EA model [25], [28], [41]. In our implementation, we adopt the state-of-the-art EA model AttrGNN [28] that aggregates the graph feature of each tuple via multiple newly proposed GNNs, for this purpose. In the following, we sketch the main idea about how to use an EA model to learn tuples' graph features in MRGFL.

The graph features of each tuple can be obtained by applying a GNN model, as described in Section 2.3. It outputs a set of tuples' embeddings. We denote the embedding of each tuple $e_i \in T$ (resp. $e'_i \in T'$) as $\mathbf{h}_{e_i}$ (resp. $\mathbf{h}_{e'_i}$). Then, a *training objective function* (denoted as $\mathcal{L}_g$) is used to unify the two datasets' tuple embeddings into a unified vector space by maximizing the similarity of each tuple pair (w.r.t. the generated positive labels). Formally,

$$\mathcal{L}_g = \sum_{(e_i,e'_i)\in\mathbb{P}} \sum_{(e_j,e_k)\in\mathbb{N}} \left[d(e_i,e'_i) + \gamma - d(e_j,e_k)\right]_+. \tag{4}$$

Here, $(e_i, e'_i) \in \mathbb{P}$ represents a positive label; $(e_j, e_k) \in \mathbb{N}$ represents a negative label; $[b]_+ = max\{0, b\}$; $d(e_i, e'_i)$ denotes the cosine distance between $\mathbf{h}_{e_i}$ and $\mathbf{h}_{e'_i}$, where $\mathbf{h}_{e_i}$ and $\mathbf{h}_{e'_i}$ are the final embeddings of e_i and e'_i w.r.t. the multi-relational graph $\mathcal{G}$ after performing the $|l|$-th layer GNN model, respectively; similarly, $d(e_j, e_k)$ represents the cosine distance between $\mathbf{h}_{e_j}$ and $\mathbf{h}_{e_k}$; and γ is a margin hyper-parameter. We set $\gamma = 1.0$ in the current implementation. According to a comprehensive survey of GNNs [47], the time complexity of GNN is $O(|E|)$, where $|E|$ denotes the number of edges $E \in \mathcal{G}$. To this end, the time complexity of the GNN-based MRGFL is $O(|\mathbb{P}| \times (|E| + |E'|))$. Here, $|\mathbb{P}|$ is the number of the generated positive labels, and $|E|$ (resp. $|E'|$) denotes the number of edges $E \in \mathcal{G}$ (resp. $E' \in \mathcal{G}'$).

5.2 Collaborative Sentence Feature Learning (CSFL)

As discussed in Section 1, treating tuples as sentences causes insufficient feature discovery. In view of this, we propose a collaborative sentence feature learning (CSFL) model, which discovers sufficient tuples' sentence features for EM with the assistance of the well-trained graph features of tuples. The training objective of CSFL is to (i) identify whether two tuples refer to the same real-world entity; and (ii) minimize the semantic distance between the matched tuples. The architecture of CSFL is depicted in Fig. 4.

First, we present how to identify the matched (or mismatched) tuples in CSFL. We fine-tune a pre-trained LM with a sentence pair classification task. We take as inputs a pairwise sentence $\mathcal{S}(e_i, e'_i)$ and its corresponding positive and negative labels generated by the proposed ALG. Then, we learn the classification signal $\mathbf{E}_{[\mathbf{CLS}]}$ by feeding the inputs into a multi-layer Transformer encoder. In the current implementation, the number of transformer layers is set to 12, a typical setting used in various tasks [26], [35]. We use a variant of *CrossEntropy Loss* $\mathcal{L}_1$ as the objective training function, which is derived from Equation (1). Formally,

$$\mathcal{L}_1(y = k|\mathcal{S}(e_i, e'_j)) = -\log\left(\frac{\exp(d^*_k)}{\sum_q^{|k|}\exp(d^*_q)}\right)\forall k \in \{0,1\} \tag{5}$$

$$d^* = \mathbf{W}_c^{*\top}(\mathbf{E}_{[\mathbf{CLS}]}; \mathbf{G}_{\mathbf{abs}}; \mathbf{G}_{\mathbf{dot}}) \tag{6}$$

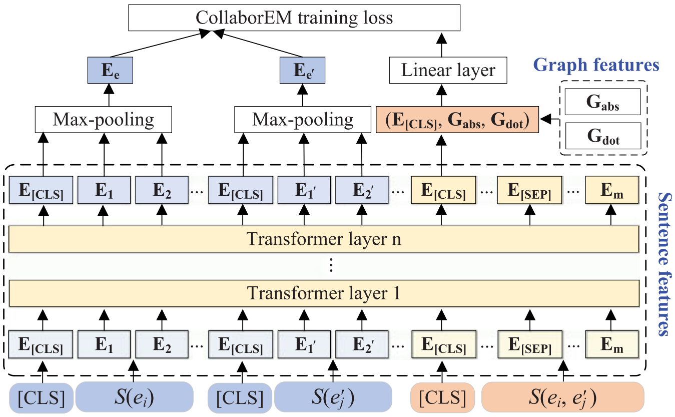

CollaborEM training loss
$\mathbf{E_e}$
$\mathbf{E_{e'}}$
Linear layer
Graph features
$\mathbf{G_{abs}}$
$\mathbf{G_{dot}}$
Max-pooling
Max-pooling
$(\mathbf{E_{[CLS]}}, \mathbf{G_{abs}}, \mathbf{G_{dot}})$
$\mathbf{E_{[CLS]}}$ $\mathbf{E_1}$ $\mathbf{E_2}$ ··· $\mathbf{E_{[CLS]}}$ $\mathbf{E_{1'}}$ $\mathbf{E_{2'}}$ ··· $\mathbf{E_{[CLS]}}$ ··· $\mathbf{E_{[SEP]}}$ ··· $\mathbf{E_m}$
Transformer layer n
Transformer layer 1
$\mathbf{E_{[CLS]}}$ $\mathbf{E_1}$ $\mathbf{E_2}$ ··· $\mathbf{E_{[CLS]}}$ $\mathbf{E_{1'}}$ $\mathbf{E_{2'}}$ ··· $\mathbf{E_{[CLS]}}$ ··· $\mathbf{E_{[SEP]}}$ ··· $\mathbf{E_m}$
[CLS] $S(e_i)$ [CLS] $S(e'_j)$ [CLS] $S(e_i, e'_j)$
Sentence features

Fig. 4. The architecture of CSFL

Here, the logits $\boldsymbol{d}^*$ is produced by both tuples' sentence features (i.e., $\mathbf{E}_{[\mathbf{CLS}]}$) and tuples' graph features (i.e., $\mathbf{G}_{\mathbf{abs}} \in \mathbb{R}^c$ and $\mathbf{G}_{\mathbf{dot}} \in \mathbb{R}^c$), where c is the dimension of the tuples' graph features. $\mathbf{W}^*_c \in \mathbb{R}^{(n+2c)\times|k|}$. $\mathbf{G}_{\mathbf{abs}} = |\mathbf{h}_{e_i} - \mathbf{h}_{e'_i}|$ denotes the element-wise difference between the graph-based embeddings $\mathbf{h}_{e_i}$ and $\mathbf{h}_{e'_i}$. $\mathbf{G}_{\mathbf{dot}} = \mathbf{h}_{e_i} \otimes \mathbf{h}_{e'_i}$ represents the element-wise similarity between $\mathbf{h}_{e_i}$ and $\mathbf{h}_{e'_i}$.

Second, we illustrate how to minimize the semantic distance between the matched tuples. At the input, the pre-trained LM allocates an initialized embedding for each token of a sentence $\mathcal{S}(e_i)$ (w.r.t. $\mathcal{S}(e'_i)$), denoted as $\mathbf{E_i}$ (w.r.t. $\mathbf{E'_i}$). Note that, the special symbol [CLS], which is located in the front of every sentence, also has an initial embedding, denoted as $\mathbf{E}_{[\mathbf{CLS}]}$. The embedding of every token will be updated after performing the multi-layer Transformer encoder. We apply *max-pooling* to obtain a fixed-length embedding $\mathbf{E}_{\mathbf{e_i}}$ (w.r.t. $\mathbf{E}_{\mathbf{e'_i}}$) for representing the tuple e (w.r.t. e'). Concretely, max-pooling generates the fixed-length embedding by selecting the maximal value in each dimension among all the embedded tokens of the tuple. We use *CosineEmbedding Loss* $\mathcal{L}_2$ as the objective training function. It is designed to minimize the semantic distance between matched tuples (w.r.t. the set of positive labels $\mathbb{P}$) and maximize that between mismatched tuples (w.r.t. the set of negative labels $\mathbb{N}$). Formally,

$$\mathcal{L}_2(y|\mathcal{S}(e_i), \mathcal{S}(e'_j)) = \begin{cases} 1 - \cos(\mathbf{E}_{\mathbf{e_i}}, \mathbf{E}_{\mathbf{e'_j}}), & \text{if } y = 1 \\ \max(0, \cos(\mathbf{E}_{\mathbf{e_i}}, \mathbf{E}_{\mathbf{e'_j}}) - \lambda), & \text{if } y = 0 \end{cases}, \tag{7}$$

where λ is a margin hyper-parameter separating matched tuple pairs from mismatched tuple pairs. $\cos(\cdot,\cdot)$ represents the cosine distance metric.

Finally, we are ready to present the overall training function of CSFL, namely, *CollaborEM training loss* (denoted as $\mathcal{L}_c$). Formally,

$$\mathcal{L}_c = \mathcal{L}_1 + \mu\mathcal{L}_2, \tag{8}$$

where hyper-parameter $\mu \in [0, 1]$ is a coefficient controlling the relative weight of $\mathcal{L}_2$ against $\mathcal{L}_1$. The EM results can be obtained according to the predicted labels of each tuple pair.

Discussion. Compared to the existing sentence-based EM methods that also fine-tune pre-trained LMs, we emphasize the superiority of CSFL in the following two aspects. First, CSFL incorporates the graph features of tuples learned in the previous MRGFL step to enrich the features that the sentence-based model fails to capture. Second, we argue that utilizing the CosineEmbedding Loss (i.e., $\mathcal{L}_2$ defined in Equation (7)) as a part of *CollaborEM training loss* is suitable for the EM task. Intuitively, matched tuples should have similar embeddings in a unified semantic vector space. However, the existing sentence-based EM methods, which finetune and cast EM as a sentence-pair classification problem, cannot ensure the semantic similarity between matched tuples. We will verify the superiority of the proposed CSFL in the experiments to be presented in Section 6.4. In addition, the time complexity of CSFL is $O(|\mathcal{S}|^2 \times d)$ according to the analysis of pre-trained LMs [20]. Here, $|S|$ is the length of the input sentence of CSFL; and d is the dimensionality of the tuple's embedding.

6 Experiments

In this section, we conduct comprehensive experiments to verify the performance of CollaborEM from three aspects. First, we compare CollaborEM with several competing EM approaches and present the results in Section 6.3. Second, we conduct the ablation study for the proposed CollaborEM and report our findings in Section 6.4. Third, we further explore CollaborEM by (i) analyzing the performance of both the reliable positive label generation (RPLG) and the similarity-based negative label generation (SNLG) (in the automatic label generation (ALG) strategy) in Section 6.5.1; and (ii) comparing the scale of the graphs generated by the proposed multi-relational graph construction (MRGC) method and other existing approaches in Section 6.5.2.

6.1 Benchmark Datasets

We conduct experiments on *eight* representative and widely-used EM benchmarks with different sizes and in various domains. Table 2 lists the detailed statistics. For structured EM, we use five benchmarks, including Amazon-Google (AG), BeerAdvo-RateBeer (BR), the clean version of DBLP-ACM (DA-clean), Fodors-Zagats (FZ), and the clean version of iTunes-Amazon (IA-clean). The attribute values of tuples are atomic but not a composition of multiple values. For dirty EM, following [33], we use the dirty versions of the DBLP-ACM (DA-dirty) and iTunes-Amazon (IA-dirty) benchmarks to measure the robustness of the proposed CollaborEM against noise. For textual EM, we use the Abt-Buy (AB) benchmark which is text-heavy, meaning that at least one attribute of each tuple contains long textual values.

TABLE 2
Statistics of the Datasets Used in Experiments

Type	Dataset	Domain	#Attr.	#Domain	#Tuple	#Pos.
Structured	AG	software	3	123 - 3,021	1,363 - 3,226	1,167
	BR	beer	4	4 - 4,343	4,345 - 3,000	68
	DA-clean	citation	4	5 - 2,507	2,616 - 2,294	2,220
	FZ	restaurant	6	16 - 533	533 - 331	110
	IA-clean	music	8	6 - 38,794	6,907 - 55,923	132
Dirty	DA-dirty	citation	4	6 - 2,588	2,616 - 2,294	2,220
	IA-dirty	music	8	7 - 55,727	6,907 - 55,923	132
Textual	AB	product	3	167 - 1,081	1,081 - 1,092	1,028

6.2 Implementation and Experimental setup

Evaluation Metric. To measure the quality of EM results, we use *F1-score*, the harmonic mean of precision (*Prec.*) and recall (*Rec.*) computed as $\frac{2\times(Prec.\times Rec.)}{(Prec.+Rec.)}$. Here, precision is defined as the fraction of match predictions that are correct; and recall is defined as the fraction of real matches being predicted as matches.

Competitors. We compare CollaborEM against 6 SOTA EM approaches. The competitors can be classified into two categories based on whether pre-defined lables are required, i.e., *unsupervised EM* and *supervised EM*.

The former refers to the group of approaches that performs EM without any label involvement, including (i) ZeroER [46], a powerful generative EM approach based on Gaussian Mixture Models for learning the match and unmatch distributions; and (ii) EMBDI [5], which automatically learns local embeddings of tuples for EM based on the attribute-centric graphs. Methods in this group are most relevant to CollaborEM.

The latter refers to the group of approaches that relies on the pre-defined labels for matching tuples, including (i) DeepMatcher+ (DM+) [26], which implements multiple EM methods and reports the best performance (highest F1-scores), including DeepER [10], Magellan [21], DeepMatcher [33], and DeepMatcher's follow-up work [13] and [18]; (ii) GraphER [24], which integrates schematic and structural information into token representations with a GNN model for EM and aggregates token-level features as the EM results; (iii) MCA [51], which incorporates attention mechanism into a sequence-based model to learn features of tuples for EM; (iv) ERGAN [36], which employs a generative adversarial network to augment labels and predict whether two entities are matched; and (v) DITTO [26], which leverages a pre-trained LM to fine-tune and cast EM as a sentence-pair classification problem. Methods in this group are used to demonstrate that the proposed CollaborEM, although not requiring any labor-intensive annotations/labels, can achieve performance that is comparable with or even better than the performance achieved by SOTA supervised EM in various real-world EM scenarios.

Note that, in the evaluation of supervised EM methods, each dataset is split into the training, validation, and test sets using the ratio of 3:1:1. For fair comparisons with supervised methods, we report the results conducted by CollaborEM on the test sets, denoted as CollaborEM − S. For fair comparisons with unsupervised methods, we report the results of CollaborEM on the whole datasets, denoted as CollaborEM − U.

Implementation details. We implemented CollaborEM[1] in PyTorch [34], the Transformers library [45], and the Sentence-Transformers library [35]. In automatic label generation (ALG), we use *stsb-roberta-base*[2] as the pre-trained LM to get the embedding for every tuple. We set $\theta = 0.03$ in the process of reliable positive label generation (RPLG) and $\epsilon = 10^3$ in the

[1] The source code of CollaborEM is available at https://github.com/ZJU-DAILY/CollaborEM

[2] https://github.com/UKPLab/sentence-transformers

[3] To avoid false negative labels, we dismiss the top-2 neighbors into consideration.

TABLE 3
Overall EM Results With and Without Any Pre-Defined Labels (F1-Score Values are in Percentage, and the Best Scores are in **bold**)

Datasets	Unsupervised		Self-supervised	Supervised					Self-supervised	
	ZeroER [46]	EMBDI [5]	CollaborEM-U	DM+ [26]	GraphER [24]	MCA [51]	ERGAN [36]	DITTO [26]	DITTO-S	CollaborEM-S
AG	48.00	63.72*	**68.61**	70.70	68.08	71.40	37.49°	**74.81***	65.74*	71.91
BR	5.76*	85.27*	**87.69**	78.80	79.71	80.00	74.42°	93.30*	90.32*	**96.55**
DA-clean	96.00	98.46*	**98.63**	98.45	96.53*	98.90	98.51°	**98.66***	98.65*	98.65
FZ	**100**	**100***	**100**	**100**	25.64*	–	98.48°	**100***	**100***	**100**
IA-clean	/	16.56*	**96.12**	91.20	\	–	77.29°	98.18*	94.34*	**100**
DA-dirty	63.20*	98.26*	**98.25**	98.10	96.06*	98.50	81.79°	98.76*	97.88*	**99.10**
IA-dirty	/	3.64*	**95.17**	79.40	\	–	67.11°	94.55*	90.00*	**98.18**
AB	54.38*	65.18*	**84.50**	62.80	31.04*	70.80	30.37°	**89.86***	85.84*	86.52

[1] *We run the publicly available source codes of "*"-marked approaches to obtain the results.*

[2] *The results of "○"-marked methods are produced by our implementation since the source code is not provided by the original paper.*

[3] *The symbol "/" indicates that the EM model **fails** to produce any result after running for 5 days in the experimental conditions.*

[4] *The symbol "\" means that the method is **NOT** able to perform the EM task in our experimental conditions due to the GPU memory limitation.*

[5] *The symbol "–" denotes that the results are not provided in the original paper. The rest are obtained from their original papers.*

process of similarity-based negative label generation (SNLG). In collaborative EM training (CEMT), the dimension of the graph feature in the process of multi-relational graph feature learning (MRGFL) is 128. Besides, in both the training and test process of CollaborEM, we apply the half-precision floating-point (fp16) optimization to save the GPU memory usage and the running time. In all experiments, the max sequence length is set to 256; the learning rate is set to 2e-5; the batch size for the AG benchmark is set to 64 while that for the other benchmarks is set to 32. The training process runs a fixed number of epochs (1, 2, 3, 6, or 30 depending on the dataset size), and returns the checkpoint at the last epoch. We set $\lambda = 0.5$ and $\mu = 0.2$ in the proposed *CollaborEM training loss*. All the experiments were conducted on a personal computer with an Intel Core i9-10900K CPU, an NVIDIA GeForce RTX3090 GPU, and 128GB memory. The programs were all implemented in Python.

6.3 Overall Performance

Table 3 summarizes the overall EM performance of CollaborEM and its competitors.

CollaborEM *versus Unsupervised Methods.* It is observed that CollaborEM significantly outperforms all the unsupervised competitors. Particularly, CollaborEM brings about 25% absolute improvement on average over the best baseline (i.e., EMBDI). The results also demonstrate that CollaborEM is more robust against data noise than ZeroER. On the dirty datasets, the performance degradation of CollaborEM is only 0.66% on average. Nevertheless, the performance of ZeroER decreases by 33%. The reason is that unsupervised methods can easily be fooled without the guidance of any supervision signal, as discussed in Section 1. On the contrary, CollaborEM generates reliable labels via the proposed ALG strategy as the supervision signals. The reliability analysis of EM labels generated by ALG can be found in Section 6.5.1. Besides, the collaborative EM training process (i.e., CEMT), which absorbs both graph features and sentence features of tuples, has the fault-tolerance capability for dealing with noisy tuples. The state-of-the-art EM performance and the robust property make CollaborEM more attractive in practical EM scenarios.

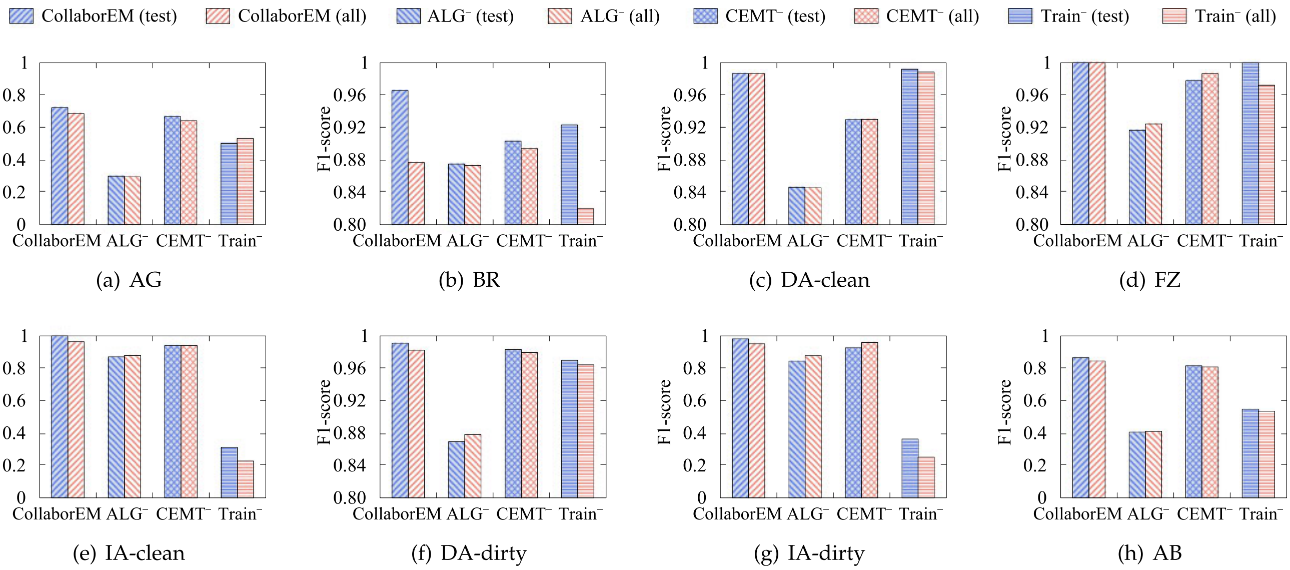
CollaborEM *versus Supervised Methods.* As we can see, the performance of CollaborEM is comparable with or even superior to the SOTA supervised EM approaches. Concretely, CollaborEM outperforms even the best supervised competitor (i.e., DITTO) by 2.26% on average over 4 datasets. Although the performance of CollaborEM in the other datasets is inferior to that of DITTO, the difference in their respective F1-scores does not exceed 3.34%. This is really impressive since CollaborEM requires *zero* human involvement in annotating labels for EM. In contrast, DITTO requires a sufficient amount of labels that are expensive to obtain and often times infeasible. To further compare the performance difference between CollaborEM and DITTO under a fair comparison, we evaluate the performance of DITTO when using the pseudo labels generated by ALG, denoted as DITTO-S. As can be observed, the performance of DITTO-S is inferior to that of CollaborEM. The reason is that, DITTO-S treats entities as sentences, and hence, it does not consider the rich semantic features of entities, as mentioned in Section 1. Nonetheless, CollaborEM has the capability to capture rich semantics of entities by discovering both sentence features and graph features of entities collaboratively.

In addition, both CollaborEM and ERGAN generate pseudo labels for improving their EM performance. However, we can observe from the results that CollaborEM achieves superior performance than ERGAN. Specifically, CollaborEM brings about 23% improvement on average on the F1-score, compared to the EM results produced by ERGAN. The inferior performance of ERGAN is attributed to the inherent GAN. Specifically, it is common that the training process of GAN is unstable [31], incurring the poor quality of the generated pseudo labels and unsatisfied training performance. By analyzing the evaluated datasets, ERGAN generates positive and negative labels with an average accuracy of 88% and 89%, respectively. However, CollaborEM produces positive and negative labels with an average accuracy of 99% and 97% respectively, as verified in Section 6.5.1. Compared to ERGAN, CollaborEM gains up to 11% improvement on the quality of the generated labels.

Efficiency Evaluation. To further investigate the efficiency of CollaborEM, we compare it with the other evaluated approaches that achieve the best quality of EM results in the corresponding categories, i.e., the unsupervised method EMBDI, the supervised approach DITTO, and the self-supervised one DITTO-S. We report the

TABLE 4
Efficiency Comparison Between CollaborEM and its Competitors (The Best Scores are in Bold)

Datasets	EMBDI				CollaborEM-U				DITTO				DITTO-S				CollaborEM-S			
	F1-score	Prec.	Rec.	Time	F1-score	Prec.	Rec.	Time	F1-score	Prec.	Rec.	Time	F1-score	Prec.	Rec.	Time	F1-score	Prec.	Rec.	Time
AG	63.72	66.49	61.18	**0.80**	68.61	63.48	74.64	4.97	**74.81**	**67.59**	**83.76**	7.61	65.74	61.57	70.51	6.25	71.91	66.55	78.21	4.97
BR	85.27	90.16	80.88	**1.06**	87.69	91.94	83.82	35.66	93.30	87.50	**100**	1.76	90.32	82.35	**100**	28.55	**96.55**	**93.33**	**100**	35.66
DA-clean	98.46	**99.04**	97.88	**1.84**	98.63	98.13	99.14	3.67	**98.66**	98.00	**99.32**	14.54	98.65	98.43	98.87	29.89	98.65	98.21	99.10	3.67
FZ	**100**	**100**	**100**	**0.48**	**100**	**100**	**100**	4.91	**100**	**100**	**100**	2.62	**100**	**100**	**100**	4.26	**100**	**100**	**100**	4.91
IA-clean	16.56	44.83	10.16	27.28	96.12	98.41	93.94	7.38	98.18	96.43	**100**	**3.35**	94.34	96.15	92.59	26.25	**100**	**100**	**100**	7.38
DA-dirty	98.26	98.86	97.66	**1.85**	98.25	98.07	98.42	6.73	98.76	98.87	98.65	16.50	97.88	96.91	98.87	30.03	**99.10**	**99.10**	**99.10**	6.73
IA-dirty	3.64	8.11	2.34	23.21	95.17	93.43	96.97	**3.83**	94.55	92.86	96.30	3.86	90.00	81.82	**100**	27.21	**98.18**	**96.43**	**100**	**3.83**
AB	65.18	**92.90**	50.20	**1.07**	84.50	81.92	87.26	10.11	**89.86**	89.42	90.29	9.68	85.84	81.03	**91.26**	7.80	86.52	84.33	88.83	10.11



(a) AG (b) BR (c) DA-clean (d) FZ

(e) IA-clean (f) DA-dirty (g) IA-dirty (h) AB

Fig. 5. Ablation study of CollaborEM. Blue bars (corresponding to test) represent the results on the test sets, and red bars (corresponding to all) denote the results on the whole datasets.

values of F1-score, precision (Prec.), recall (Rec.), and training time in Table 4. The values of training time are in minutes. The other values are in percentage. In addition, we did not report the test time of every evaluated approach. This is because, it is common for entity matching methods to use the same strategy for evaluating the EM results in the test set, resulting in the similar test time of different approaches.

As observed, compared to the best unsupervised method (i.e., EMBDI), CollaborEM achieves superior quality of EM results. Especially on the IA-dirty dataset, CollaborEM outperforms EMBDI by more than 85% on both F1-score, precision, and recall. However, CollaborEM needs more training time to obtain the state-of-the-art EM results, compared with EMBDI. This is because, the collaborative training model of CollaborEM (i.e., CEMT) contains a pre-trained language model, which is much more complex in the model structure, and needs to store more parameters in the training process. EMBDI relies on the random-walk model, which is relatively lightweight in the model structure, and requires less training parameters. This demonstrates a trade-off between the effectiveness and the efficiency of EM problem. To sum up, it is significant that spending a relatively longer time in getting better entity matching results.

It is also observed that, compared against the state-of-the-art supervised method (DITTO), CollaborEM achieves comparable or even superior performance in F1-score, precision, recall, and training time. This confirms that i) CollaborEM is able to generate pseudo labels with high quality; and ii) CollaborEM can discover more sufficient tuple features by collaboratively using both the graph feature and sentence feature of tuples, compared to DITTO that treats each tuple as a sentence. In addition, compared to the self-supervised variant of DITTO (i.e., DITTO-S) that also employs the pseudo labels generated by ALG, CollaborEM shows higher quality of results and faster training time in most cases. This further demonstrates that CollaborEM is effective and efficient in solving the EM problem.

6.4 Ablation Study

Next, we analyse the effectiveness of each proposed phase of CollaborEM (i.e., ALG and CEMT) by comparing CollaborEM with its variants without the key optimization (s) in each phase. The results are shown in Fig. 5, where the labels listed along the abscissa have the following meanings: (i) "CollaborEM" represents its performance when all optimizations are used; (ii) "ALG$^-$" means the performance of CollaborEM without (w/o) ALG; (iii) "CEMT$^-$" denotes the performance of CollaborEM w/o CEMT; and (iv) "Train$^-$" represents the performance of CollaborEM w/o training.

CollaborEM *versus* CollaborEM *w/o ALG.* ALG contains two components, i.e., RPLG and SNLG. Since CollaborEM cannot work without RPLG, we focus on investigating the effectiveness of SNLG by replacing it with a random negative label generation method. It is observed that the F1-score drops 18.04% on average. This confirms that generating "challenging" negative labels based on semantic similarity greatly helps to train effective EM models. We also observe that the SNLG brings no more than 8.33% improvement on FZ dataset. This is attributed to the nature of this dataset, as it is relatively easier for CollaborEM and all the competitors to achieve the perfect performance, i.e., 100% F1-score.

CollaborEM *versus* CollaborEM *w/o CEMT.* The difference between the proposed CEMT and other existing EM models that also fine-tune pre-trained LMs lies in whether there is the intervention of graph features (w.r.t MRGFL) to assist the fine-tuning process. By removing MRGFL in CEMT, the F1-score of CollaborEM drops 3% on average over the eight experimental datasets. Particularly, the drop of the F1-score is up to 5% on the DA-clean dataset. This shows that learning tuples' graph features is indispensable for promoting EM performance.

CollaborEM *versus* CollaborEM *w/o Train.* We also explore the performance of CollaborEM without any training process. In this case, CollaborEM performs EM purely based on RPLG, which automatically discovers the matched tuples based on the semantic similarity. The results indicate that RPLG can find a large quantity of reliable matched tuples. It is worth noting that RPLG alone can achieve considerable results, e.g., $\sim$ 99% F1-score and 100% F1-score on DA-clean dataset and FZ dataset, respectively. This is because, matched tuples are mutually most similar with each other in those datasets. Since RPLG is general enough to perform EM in various datasets, it is possible to be widely used in practical EM applications without any time-consuming training process.

6.5 Further Experiments

We further justify the effectiveness of the proposed CollaborEM by conducting the following two sets of experiments.

6.5.1 ALG Analysis

The first set of experiments is to verify the performance of ALG. To better study the quality of labels, we utilize six metrics: (i) *true-positive* (TP), which represents the number of truly labeled matched tuples; (ii) *true-negative* (TN), which denotes the number of truly labeled mismatched tuples; (iii) *false-negative* (FN), which represents the number of matched tuples that are labeled as mismatched; (iv) *false-positive* (FP), which denotes the number of mismatched tuples that are labeled as matched; (v) *true-positive rate* (TPR) represents the proportion of matched tuples that are correctly labeled, denoted as $\frac{TP}{TP+FN}$; and (vi) *true-negative rate* (TNR) represents the proportion of mismatched tuples that are correctly labeled, denoted as $\frac{TN}{TN+FP}$.

Analysis of Label Generating Quality. We first evaluate the quality of the labels generated by ALG, including the positive labels generated by PRLG and the negative labels produced by SNLG. The results are reported in Table 5. As expected, both PRLG and SNLG are able to achieve the state-of-the-art performance when generating labels. Specifically, CollaborEM produces positive and negative labels with an average accuracy of 99% and 97%, respectively. It confirms the effectiveness of our proposed ALG. The positive labels with high reliability allow the subsequent CEMT model to be well-trained; while the generated negative labels enable CEMT to identify "challenging" tuple pairs.

TABLE 5
The Reliability Analysis of ALG

Datasets	RPLG			SNLG		
	TP	FN	TPR	TN	FP	TNR
AG	332	0	1	7136	115	0.98
BR	34	0	1	17417	1074	0.94
DA-clean	2136	0	1	34119	3	0.99
FZ	102	0	1	1788	11	0.99
IA-clean	4	0	1	8399	521	0.94
DA-dirty	1941	0	1	30899	7	0.99
IA-dirty	2	0	1	7872	490	0.94
AB	247	2	0.99	4093	11	0.99

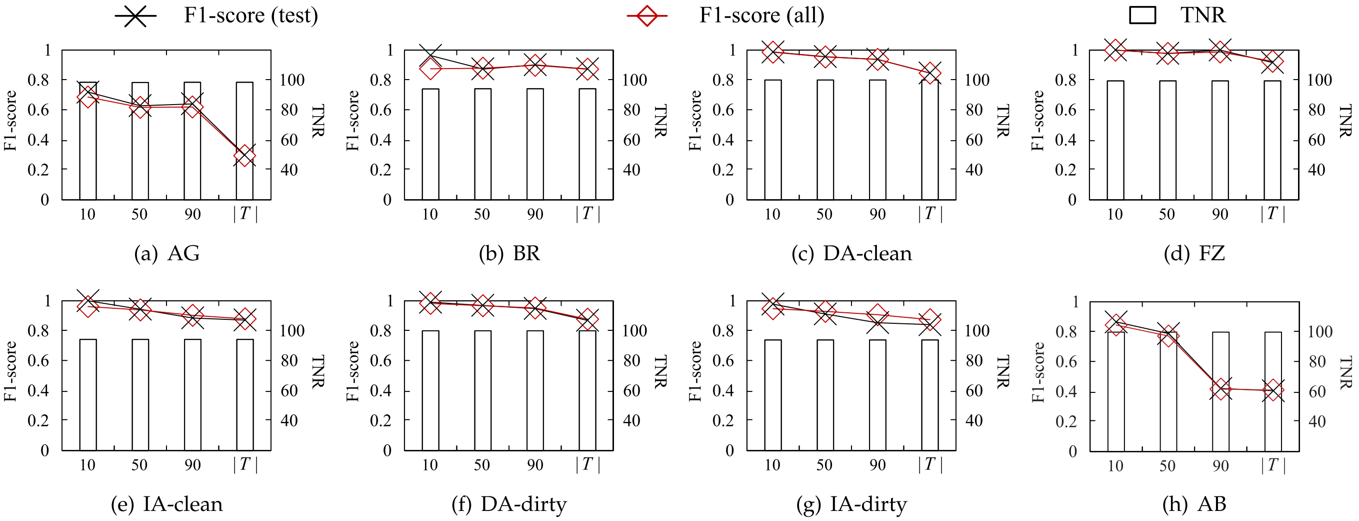
Effect of ϵ-Nearest Neighbors for SNLG. We then study the performance of SNLG by varying ϵ among {10, 50, 90, $|T|$}. Note that, when $\epsilon = |T|$, SNLG generates negative labels by searching for possible tuples in the entire dataset. In this case, SNLG behaves like random sampling. Fig. 6 plots the corresponding results. We observe that F1-score of CollaborEM drops as ϵ grows. This is because, the larger the ϵ, the more likely the tuple pairs that are not similar to each other will be included in the set of negative labels. The dissimilar tuples contribute little to the training of an effective EM model, as discussed in Section 3. Besides, as expected, the quality of negative labels is still stable when ϵ changes, which could be observed from TNR (true-negative rate) values. This further demonstrates the effectiveness of the proposed SNLG.

6.5.2 Graph Scale Analysis

The second set of experiments is to compare the scale of the graphs generated by the proposed MRGC and other graph construction methods in the existing EM approaches, i.e., EMBDI [5] and GraphER [24]. Fig. 7 depicts the total number of nodes (denoted as #Nodes) and that of edges (denoted as #Edges) of graphs with regard to each dataset. It is observed that MRGC generates much smaller graphs, compared against other graph generation methods. The reduced size of graphs greatly saves the memory for storing and training graphs, and reduces the training cost.

7 Related Work

Entity Matching (EM) is one of the fundamental and significant tasks in data curation. Early studies exploit rules [1], [12], [17], [37], [38] or crowdsourcing [14], [30], [44] for EM tasks. Rule-based solutions require human-provided declarative matching rules [17] or program-synthesized matching rules [38] to find matching pairs. Crowdsourcing-based solutions employ crowds to manually identify whether two tuples refer to the same real-world entity. Such solutions highly rely on human guidance, and have limitations in handling heterogeneous data. Recently, machine learning (ML) techniques have been widely used for EM and have



(a) AG (b) BR (c) DA-clean (d) FZ

(e) IA-clean (f) DA-dirty (g) IA-dirty (h) AB

Fig. 6. Effect of ϵ-nearest neighbors for SNLG.

achieved promising performance [9]. According to whether supervision signals are incorporated, existing ML-based solutions can be clustered into two categories, namely, *supervised EM* and *unsupervised EM*.

Supervised EM approaches [4], [5], [10], [13], [18], [22], [23], [24], [26], [33], [51], [52] can provide the state-of-the-art performance for EM, but require a substantial number of labels in the form of matches and mismatches, to support the learning of a reliable EM model. In general, the methods first learn the features of tuples via ML models and then feed the well-trained features into a binary classifier for identifying matched tuples.

A majority of supervised EM methods employ sentence-based ML model to learn the *sentence features* of tuples. DeepER [10] and DeepMatcher [33] utilize vanilla RNNs. MCA [51] proposes a multi-context attention mechanism to enrich the sentence features of tuples. Furthermore, current studies [4], [23], [26] indicate that applying pre-trained LMs to EM tasks achieves the state-of-the-art performance. DITTO [26] obtains the best performance among all the existing supervised EM works. It fine-tunes the pre-trained LMs with the help of a series of newly proposed data augmentation techniques. Several supervised EM methods transform a collection of tuples with the relational format to graph structures, and learn the *graph features* of tuples based on the constructed graphs [5], [24].

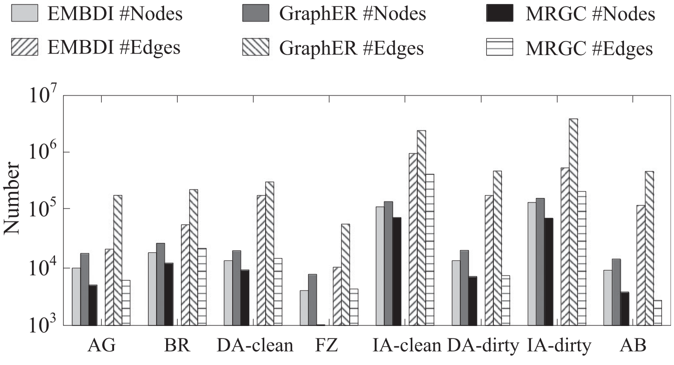


Fig. 7. Graph scale analysis.

However, both sentence-based methods and graph-based methods are far from enough to capture sufficient features of tuples, as mentioned in Section 1. Our proposed CollaborEM is introduced to enrich the features of tuples by learning both sentence features and graph features collaboratively. Besides, we have compared CollaborEM with *four* state-of-the-art supervised EM solutions, and have verified that CollaborEM, with zero labor-intensive labeling process, achieves comparable or even superior results, as compared with supervised approaches.

Unsupervised EM approaches [5], [46], [50] are designed to perform EM without labeling. ZeroER [46] learns the match and mismatch distributions based on Gaussian Mixture Models. EMBDI [5] performs EM by learning a compact graph-based representation for each tuple. ITER+CliqueRank [50] first constructs a bipartite graph to model the relationship between tuple pairs, and then develops an iterative-based ranking algorithm to estimate the similarity of tuple pairs. Despite the benefit of zero label requirement, unsupervised approaches are highly error-sensitive and may suffer from poor EM results when errors are contained in datasets. Considering that real-world datasets are often dirty, it is impractical to use the existing unsupervised EM methods in practice.

On the contrary, the proposed CollaborEM performs EM in a self-supervised manner, which has the capability to perform EM in a fault-tolerant manner, as verified in the experiments reported in Section 6.3. In addition, we have compared CollaborEM with two state-of-the-art unsupervised methods, including ZeroER and EMBDI. Note that we exclude ITER+CliqueRank from experiments since its performance is inferior to the two unsupervised methods that are selected as competitors in our study.

8 Conclusion

In this paper, we propose CollaborEM, a self-supervised entity resolution framework, to perform the EM task with *zero* labor-intensive manual labeling. CollaborEM conducts EM tasks by a pipe-lined modular architecture consisting of two phases, i.e., *automatic label generation* (ALG) and *collaborative*

EM training (CEMT). First, ALG is developed to automatically generate both *reliable positive labels* (w.r.t. RPLG) and *semantic-based negative labels* (w.r.t. SNLG). ALG is essential for the subsequent CEMT phase since it provides high-quality labels that are the backbone of training effective EM models. Second, the framework proceeds to the CEMT phase, where tuples' sentence features and graph features are learned and employed collaboratively to produce the final EM results. In this phase, we first propose a *multi-relational graph construction* (MRGC) method to construct graphs for each relational dataset, and then exploit GNN to learn the graph features of tuples. Thereafter, the well-trained graph features are fed into *a collaborative sentence feature learning* (CSFL) model to discover sufficient sentence features of tuples. Finally, CSFL predicts the matched tuple pairs and unmatched ones according to the learned features.

Recall that the blocking phase of EM is able to reduce the quadratic number of candidates of matched tuple pairs. To enable CollaborEM to be scalable, we plan to explore an effective and efficient blocking method in the near future.

References

[1] A. Arasu, C. Ré, and D. Suciu, "Large-scale deduplication with constraints using dedupalog," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 952–963.

[2] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: Unsupervised anomaly detection on multivariate time series," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2020, pp. 3395–3404.

[3] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 695–704.

[4] U. Brunner and K. Stockinger, "Entity matching with transformer architectures - A step forward in data integration," in *Proc. Int. Conf. Extending Database Technol.*, 2020, pp. 463–473.

[5] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1335–1349.

[6] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 941–949.

[7] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Berlin, Germany: Springer, 2012.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics-Hum. Lang. Technologies*, 2019, pp. 4171–4186.

[9] X. L. Dong and T. Rekatsinas, "Data integration and machine learning: A natural synergy," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 1645–1650.

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

[11] W. Fan and F. Geerts, *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. San Rafael, CA, USA: Morgan & Claypool, 2012.

[12] W. Fan, X. Jia, J. Li, and S. Ma, "Reasoning about record matching rules," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 407–418, 2009.

[13] C. Fu *et al.*, "End-to-end multi-perspective matching for entity resolution," in *Proc. 28th Int. Joint Conf. Artif. Intell. Main Track*, 2019, pp. 4961–4967.

[14] C. Gokhale *et al.*, "Corleone: Hands-off crowdsourcing for entity matching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 601–612.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Representations*, 2015.

[16] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[17] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1995, pp. 127–138.

[18] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, "Low-resource deep entity resolution with transfer and active learning," in *Proc. Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 5851–5861.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[20] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proc. Int. Conf. Learn. Representations*, 2020.

[21] P. Konda *et al.*, "Magellan: Toward building entity matching management systems," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1197–1208, 2016.

[22] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," *Proc. VLDB Endow.*, vol. 3, no. 1, pp. 484–493, 2010.

[23] B. Li, Y. Miao, Y. Wang, Y. Sun, and W. Wang, "Improving the efficiency and effectiveness for bert-based entity resolution," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 13226–13233.

[24] B. Li, W. Wang, Y. Sun, L. Zhang, M. A. Ali, and Y. Wang, "Grapher: Token-centric entity resolution with graph convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 8172–8179.

[25] C. Li, Y. Cao, L. Hou, J. Shi, J. Li, and T. Chua, "Semi-supervised entity alignment via joint knowledge embedding model and cross-graph model," in *Proc. Conf. Empir. Methods Natural Lang. Natural Lang. Process.*, 2019, pp. 2723–2732.

[26] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, "Deep entity matching with pre-trained language models," *Proc. VLDB Endow.*, vol. 14, no. 1, pp. 50–60, 2020.

[27] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi, "Deep text classification can be fooled," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 4208–4215.

[28] Z. Liu, Y. Cao, L. Pan, J. Li, and T. Chua, "Exploring and evaluating attributes, values, and structures for entity alignment," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2020, pp. 6355–6364.

[29] X. Mao, W. Wang, H. Xu, Y. Wu, and M. Lan, "Relational reflection entity alignment," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, 2020, pp. 1095–1104.

[30] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Human-powered sorts and joins," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 13–24, 2011.

[31] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[32] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[33] S. Mudgal *et al.*, "Deep learning for entity matching: A design space exploration," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 19–34.

[34] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[35] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2019, pp. 3980–3990.

[36] J. Shao, Q. Wang, A. Wijesinghe, and E. Rahm, "ERGAN: Generative adversarial networks for entity resolution," in *Proc. IEEE Int. Conf. Data Mining*, 2020, pp. 1250–1255.

[37] R. Singh *et al.*, "Generating concise entity matching rules," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 1635–1638.

[38] R. Singh *et al.*, "Synthesizing entity matching rules by examples," *Proc. VLDB Endow.*, vol. 11, no. 2, pp. 189–202, 2017.

[39] Z. Sun, Z. Deng, J. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *Proc. Int. Conf. Learn. Representations*, 2019.

[40] Z. Sun, W. Hu, Q. Zhang, and Y. Qu, "Bootstrapping entity alignment with knowledge graph embedding," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 4396–4402.

[41] Z. Sun *et al.*, "A benchmarking study of embedding-based entity alignment for knowledge graphs," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2326–2340, 2020.

[42] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2071–2080.

[43] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[44] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "Crowder: Crowdsourcing entity resolution," *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1483–1494, 2012.

[45] T. Wolf *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," 2019, *arXiv:1910.03771*.

[46] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan, "Zeroer: Entity resolution using zero labeled examples," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1149–1164.

[47] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, 32, no. 1, pp. 4—24, Jan. 2021.

[48] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 5754–5764.

[49] W. Zeng, X. Zhao, W. Wang, J. Tang, and Z. Tan, "Degree-aware alignment for entities in tail," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 811–820.

[50] D. Zhang, D. Li, L. Guo, and K.-L. Tan, "Unsupervised entity resolution with blocking and graph algorithms," *IEEE Trans. Knowl. Data Eng.*, early access, Apr. 28, 2020, doi: 10.1109/TKDE.2020.2991063.

[51] D. Zhang, Y. Nie, S. Wu, Y. Shen, and K. Tan, "Multi-context attention for entity matching," in *Proc. Web Conf.*, 2020, pp. 2634–2640.

[52] C. Zhao and Y. He, "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning," in *Proc. World Wide Web Conf.*, 2019, pp. 2413–2424.

[53] B. Zong *et al.*, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Representations*, 2018.

Congcong Ge received the BS degree in computer science from the Zhejiang University of Technology, China, in 2017. She is currently working toward the PhD degree at the College of Computer Science, Zhejiang University, China. Her research interests include data cleaning and data integration.

Pengfei Wang received the BS degree in computer science from the Zhejiang University of Technology, China, in 2021. He is currently working toward the master's degree at the School of Software, Zhejiang University, China. His research interests include data cleaning and data integration.

Lu Chen received the PhD degree in computer science from Zhejiang University, China, in 2016. She was an assistant professor with Aalborg University for a two year period from 2017 to 2019, and she was an associate professor with Aalborg University for a one year period from 2019 to 2020. She is currently a ZJU Plan 100 professor with the College of Computer Science, Zhejiang University, Hangzhou, China. Her research interests include indexing and querying metric spaces, graph databases, and database usability.

Xiaoze Liu received the BS degree in IoT Engineering from Northeastern University, China, in 2020. He is currently working toward the master's degree at the College of Computer Science, Zhejiang University, China. His research interests include knowledge graphs and data integration.

Baihua Zheng received the PhD degree in computer science from the Hong Kong University of Science & Technology, China, in 2003. She is currently a professor with the School of Computing and Information Systems, Singapore Management University, Singapore. Her research interests include mobile/pervasive computing, spatial databases, and big data analytics.

Yunjun Gao (Member, IEEE) received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently a professor with the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, graph databases, spatio-textual data processing, and database usability. He is a member of the ACM.