PromptEM: Prompt-tuning for Low-resource Generalized Entity Matching

Pengfei Wang
Zhejiang University
wangpf@zju.edu.cn

Xiaocan Zeng
Zhejiang University
zengxc@zju.edu.cn

Lu Chen, Fan Ye
Zhejiang University
{luchen,fan.ye}@zju.edu.cn

Yuren Mao
Zhejiang University
yuren.mao@zju.edu.cn

Junhao Zhu
Zhejiang University
zhujunhao@zju.edu.cn

Yunjun Gao
Zhejiang University
gaoyj@zju.edu.cn

ABSTRACT

Entity Matching (EM), which aims to identify whether two entity records from two relational tables refer to the same real-world entity, is one of the fundamental problems in data management. Traditional EM assumes that two tables are homogeneous with the aligned schema, while it is common that entity records of different formats (e.g., relational, semi-structured, or textual types) involve in practical scenarios. It is not practical to unify their schemas due to the different formats. To support EM on format-different entity records, Generalized Entity Matching (GEM) has been proposed and gained much attention recently. To do GEM, existing methods typically perform in a supervised learning way, which relies on a large amount of high-quality labeled examples. However, the labeling process is extremely labor-intensive, and frustrates the use of GEM. Low-resource GEM, i.e., GEM that only requires a small number of labeled examples, becomes an urgent need. To this end, this paper, for the first time, focuses on the low-resource GEM and proposes a novel low-resource GEM method, termed as PromptEM. PromptEM has addressed three challenging issues (i.e., designing GEM-specific prompt-tuning, improving pseudo-labels quality, and running efficient self-training) in low-resource GEM. Extensive experimental results on eight real benchmarks demonstrate the superiority of PromptEM in terms of effectiveness and efficiency.

1 INTRODUCTION

Entity Matching (EM) aims to identify whether two entity records from two relational tables refer to the same real-world entity, which is one of the fundamental and significant tasks in data management.
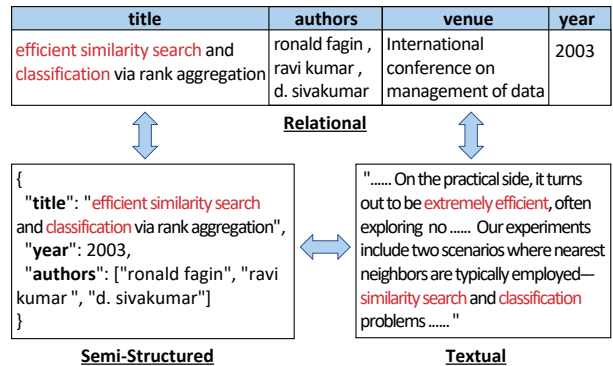
Figure 1: An example of generalized entity matching.

Most existing solutions [24, 31, 32] assume that two tables are homogeneous with the aligned schema. However, there is an urgent need to generalize the entity matching problem for more practical scenarios. Taking paper matching as an example, as shown in Figure 1, paper metadata is usually stored in relational tables or semi-structured JSON objects, while paper description (e.g., abstract) is textual data. It is not practical to unify their schemas since we need a potentially expensive schema matching in the pre-processing step [29], which is even not applicable when matching data of different formats (e.g., matching paper metadata stored in tabular format with their textual abstracts). Therefore, traditional EM is unable to support those practical scenarios like paper matching [53]. Recently, TDmatch [1] first attempts to match structured data and textual data while having extremely poor scalability (to be analyzed via experiments in Section 5.4). To support more practical application scenarios, following the previous study [53], we study *Generalized Entity Matching* (GEM), which allows matching entity records of relational, semi-structured, or textual types.

Existing methods designed for GEM typically perform in a supervised learning way, which relies on a large amount of labeled training data, and thus, is extremely labor-intensive. Recent studies [24, 31] have achieved considerable performance by leveraging the power of pre-trained language models (LMs) and the fine-tuning paradigm. Nonetheless, the fine-tuning paradigm still requires a non-trivial amount of high-quality labeled examples (e.g., thousands of labels for a typical GEM application [53]). TDmatch [1] performs in an unsupervised learning way via graph creation and random walk. However, two drawbacks restrict its real-world applications as confirmed via experiments in Section 5: (i) it has unstable performance due to the absence of labeled samples; and (ii) random

walk is not scalable on large datasets [47], incurring enormous execution time and memory usage (e.g., more than 120 hours and 130 gigabytes for once training of the SEMI-REL). To satisfy real-life applications, a low-resource (i.e., using only a small number of labeled examples) but an effective and efficient solution is required. To the best of our knowledge, low-resource GEM remains unexplored.

To overcome the low-resource dilemma, semi-supervised learning techniques (e.g., self-training [49]) are good choice. Self-training has recently been shown to obtain state-of-the-art performance for low-resource tasks like sequence generation [20] and speech recognition [55]. In self-training, a *teacher* model is trained on some labeled data, and is used to produce pseudo-labels on unlabeled data. Furthermore, the original labeled data is augmented with the pseudo-label data, and is employed to train a *student* model. Thus, a large amount of unlabeled data can be utilized effectively and the requirement of labeled data is correspondingly reduced. Although self-training has achieved promising performance in a wide range of applications, it has not been explored in GEM.

Motivated by the above considerations, we study the problem of learning high-quality models for low-resource GEM by means of self-training. Our goal is to develop an effective and efficient low-resource GEM solution based on pre-trained LMs along with leveraging self-training to boost performance, which is a challenging endeavor. The challenges are mainly three-folds:

Challenge I: *How to tune pre-trained LMs for GEM better?* Despite the success of fine-tuning LMs for the matching problem, some recent studies [18, 25, 26] find that there is a significant gap between objective forms in pre-training and fine-tuning, which restricts taking full advantage of knowledge in LMs. Pre-training is usually formalized as a cloze-style task to predict target words (e.g., masked language models). However, the approaches based on fine-tuning add additional layers to do different objective forms (e.g., classification and generation) as illustrated in Figure 2. To do GEM, existing approaches treat it as a classification problem. This gap hinders the transfer and adaptation of knowledge in LMs for GEM tasks.

Challenge II: *How to select high-quality pseudo-labels?* The quality of pseudo-labels determines whether self-training can improve performance. Thus, the pseudo-label selection strategy is extremely important. A common strategy is using confidence to select pseudo-labels. However, this strategy has some serious drawbacks [33, 39]. On the one hand, incorrect predictions can have high confidence scores in poorly calibrated networks. On the other hand, if we only aim at the pseudo-labels with high confidence produced by the teacher, there is little to gain for the student model.

Challenge III: *How to avoid expensive self-training?* Traditional self-training can be costly. To be more specific, the labeled data is augmented by the pseudo-labels produced by the teacher model, which may be beneficial to performance but result in a longer training time. Intuitively, maybe not all training data contribute to boosting the performance of the student model. Nevertheless, how to quantify the importance of training data to avoid expensive self-training is still challenging.

To tackle the above three challenges, we propose a low-resource GEM solution PromptEM. Prompt-tuning is a new promising paradigm in natural language processing, and is able to bridge the gap of objective forms between pre-training and fine-tuning [25, 41]. To address the gap between pre-training and fine-tuning (**CI**), we

cast GEM as a cloze-style task via designing the GEM-specific prompt-tuning, which has the same objective form as pre-training. Thus, we can stimulate the rich knowledge distributed in LMs through prompt-tuning. To select high-quality pseudo-labels (**CII**), we develop a lightweight uncertainty-aware self-training method to boost performance. High-quality pseudo-labels are a prerequisite for boosting performance of self-training. To this end, we employ recent advances in Bayesian deep learning [12] to obtain uncertainty estimates of the teacher model for pseudo-labeling and boosting the self-training process. To avoid expensive self-training (**CIII**), we prune useless training data dynamically using our proposed MC-EL2N, making the self-training process more lightweight and efficient. Our contributions are summarized as follows:

- *Low-resource GEM.* This is the first work formally studying the problem of low-resource generalized entity matching. We articulate the importance of this problem in more practical scenarios.
- *Prompt-tuning for GEM.* We present PromptEM, a new GEM solution based on prompt-tuning, which casts GEM as a cloze-style task. To the best of our knowledge, PromptEM is the first GEM (EM) solution that stimulates the rich knowledge distributed in LMs via designing GEM-specific prompt-tuning.
- *Generic Lightweight Self-training.* To further improve the performance in low-resource settings, we develop a generic lightweight self-training method, which selects pseudo-labels using uncertainty and makes self-training more lightweight and efficient by dynamic data pruning.
- *Extensive Experiments.* We conduct comprehensive experimental evaluation on GEM tasks compared against state-of-the-art approaches, using eight real-world datasets from various domains. Extensive experimental results demonstrate the superiority of our proposed PromptEM in terms of effectiveness and efficiency.

Outline. Section 2 presents the problem definition and overviews preliminaries. We introduce prompt-tuning for GEM in Section 3. We further improve the performance by lightweight self-training in Section 4. Section 5 presents the experimental results. Finally, we discuss related work in Section 6 and conclude in Section 7.

2 PRELIMINARIES

In this section, we first present the problem definition of generalized entity matching (GEM). Next, we introduce the serializing method for GEM, followed by an introduction of conventional vanilla fine-tuning and prompt-based tuning with LMs.

2.1 Problem Formulation

Given two collections of data entries, entity matching (EM) is to identify pairs of data entries that refer to the same real-world entity. A classic EM workflow [23] has two main steps: blocking and matching. The blocking [45] typically uses simple heuristics or deep learning techniques to reduce the quadratic number of candidates. The matching identifies whether each candidate pair is a real match or not. In this paper, we focus on the matching. Formally, given two tables E_A and E_B, we assign a binary label $y \in \{0, 1\}$ for each candidate pair $(e_a, e_b) \in E_A \times E_B$. Here, $y = 1$ denotes a truly matched pair, while $y = 0$ represents a mismatched pair.

To generalize the classic setting to more practical application scenarios, Machamp [53] comes up with the new research problem,
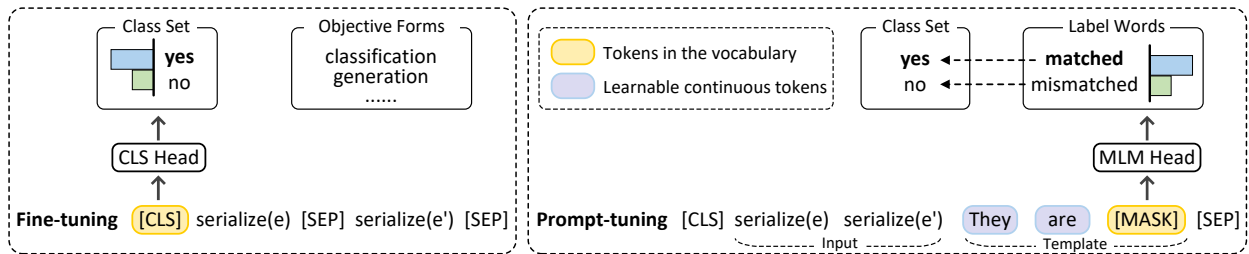
Figure 2: The illustration of fine-tuning and prompt-tuning. The blue rectangles in the figure are special prompt tokens, whose parameters are initialized and learnable during prompt-tuning.

generalized entity matching (GEM). GEM can support a variety of matching tasks with practical applications.

PROBLEM 1. *Generalized Entity Matching (GEM). Given two structured, semi-structured, or unstructured entity tables E_A and E_B with homogeneous or heterogeneous schema, GEM is to assign a binary label $y \in \{0, 1\}$ for each candidate $(e_a, e_b) \in E_A \times E_B$.*

2.2 Serializing

The matching problem can be effectively solved by formulating it as a sequence classification task [15, 24, 53]. First, entity pairs are serialized to sequences, and then, a pre-trained LM is fine-tuned to solve the task. Existing methods are designed for EM over structured data with homogeneous data, which are not suitable for GEM. Following [53], we extend the serialization method presented in Ditto [24] and introduce a reasonable way to fulfill this task.
Structured tables. For structured tables, an entity with n attributes can be denoted as $e = \{attr_i, val_i\}_{i \in [1,n]}$, where $attr_i$ is the attribute name and val_i is the corresponding attribute value. Then the serialization is denoted as:
$serialize(e) ::= [COL] attr_1 [VAL] val_1 \ldots [COL] attr_n [VAL] val_n$
where [COL] and [VAL] are two special tags indicating the start of attribute names and values respectively. Taking the relational entity in Figure 1 as an example, we serialize it as:
[COL] title [VAL] efficient similarity . . . [COL] authors [VAL] renald . . . [COL] venue [VAL] SIGMOD [COL] year [VAL] 2003
Semi-structured tables. The semi-structured tables can be serialized in a similar way. Specially, two differences exist: (i) For nested attributes, we recursively add the [COL] and [VAL] tags along with attribute names and values in each level of nests. (ii) To reduce the length of the sequence while ensuring the amount of information, we concatenate the elements in the list into one string for attributes whose content is a list. As an example, given the semi-structured entity in Figure 1, we serialize it as:
[COL] title [VAL] efficient similarity . . . [COL] year [VAL] 2003 [COL] authors [VAL] ronald fagin ravi kumar d. sivakumar
Unstructured tables. Unstructured textual entities are sequences originally, and hence, there does not need to serialize them.

2.3 Vanilla Fine-tuning

Pre-trained language models (LMs) (e.g., BERT [6] and RoBERTa [27]) have demonstrated powerful semantic expression abilities, which can support lots of downstream tasks (e.g., classification and question answering). Formally, GEM can be treated as a sequence pair classification task denoted as $\mathcal{T} = \{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X}$ is a candidate

pair set and $\mathcal{Y}$ is a class set. For each instance $x \in \mathcal{X}$, it is serialized by $x ::= [CLS] serialize(e)[SEP] serialize(e')[SEP]$, and is annotated with a label $y \in \mathcal{Y}$. Here, [CLS] and [SEP] are special tokens used to mark the beginning and end of a sequence.

Given a LM $\mathcal{M}$, vanilla fine-tuning first coverts x to the input sequence $\{[CLS], w_1, \ldots, w_n, [SEP], w'_1, \ldots, w'_m, [SEP]\}$, and then, it uses $\mathcal{M}$ to encode all tokens of the input sequence into corresponding vectors $\left\{ \mathbf{h}_{[CLS]}, \mathbf{h}_{w_1}, \ldots, \mathbf{h}_{w_n}, \mathbf{h}_{[SEP]}, \mathbf{h}_{w'_1}, \ldots, \mathbf{h}_{w'_m}, \mathbf{h}_{[SEP]} \right\}$, where w_i is the token and $\mathbf{h}_{w_i}$ is the corresponding embedding. For a downstream classification task (e.g., GEM), a task-specific head is trained to predict the probability distribution over the label set y with the softmax function $p(\cdot \mid x) = \text{Softmax} \left(\mathbf{W} \times \mathbf{h}_{[CLS]} + \mathbf{b} \right)$. Here, $\mathbf{h}_{[CLS]}$ is the embedding of special classification token [CLS], b is the bias for the layer, and $\mathbf{W}$ is a learnable matrix randomly initialized before fine-tuning. The parameters of $\mathcal{M}$, b, and $\mathbf{W}$ are tuned to maximize $\frac{1}{|X|} \sum_{x \in \mathcal{X}} \log p(y \mid x)$.

2.4 Prompt-based Tuning

Prompt-based tuning has been proposed to apply cloze-style tasks to tune LMs. Formally, we define a label word set $\mathcal{V}_y = \{w_1, \ldots, w_m\}$. $\mathcal{V}_y$ is a subset of the vocabulary $\mathcal{V}$ of the LM $\mathcal{M}$, i.e., $\mathcal{V}_y \subseteq \mathcal{V}$. We get an overall dictionary $\mathcal{V}^*$ by taking the union of the dictionary corresponding to each label. Another primary component of prompt-tuning is a prompt template $T(\cdot)$, which modifies the original input x into a prompt input $T(x)$ by adding a set of additional tokens in x. Generally, a token [MASK] is added for LMs to predict the missing label word $w \in \mathcal{V}^*$. Thus, in prompt-tuning, a classification problem is transferred into a masked language modeling problem $p(y \in \mathcal{Y} \mid x) = p([MASK] = w \in \mathcal{V}_y \mid T(x))$.

3 PROMPT TUNING FOR GEM

In this section, we detail how to utilize prompt-tuning to deal with GEM. We first design GEM-specific prompt templates and label words, and then, we describe the training and inference process.

3.1 Prompt Templates

To cast the GEM problem as a prompt-tuning one, we first design suitable prompt templates (i.e., hard-encoding and continuous templates) and label words set (to consider general binary relationship).
Hard-encoding templates. For the choice of hard-encoding templates, we do not use automatic searching methods for discrete prompts since the GEM task is clearly defined and the prompts are easily purposeful. Given each candidate pair $x = (e, e')$, we construct the following templates:

$T_1(x) = serialize(e) \; serialize(e')$ They are [MASK]

$T_2(x) = serialize(e)$ is [MASK] to $serialize(e')$

Continuous templates. As prompt construction is to find a method that allows a LM to effectively perform a task, rather than being for human consumption, it is not necessary to limit the prompt to human-interpretable natural language [25]. Thus, continuous prompts are proposed to perform prompting directly in the embedding space of the model. Here, we employ P-tuning [26], where continuous prompt tokens are learned by inserting trainable variables into the embedding input. Specifically, trainable prompt tokens are initialized, and then, BiLSTM [17] is utilized to account for interaction between prompt tokens. This enables the model to find better continuous prompts beyond the original vocabulary $\mathcal{V}$ of $\mathcal{M}$ could express. We give an illustrative example in Figure 2.

Label words set. In addition to designing templates, another primary component is to design the label words set. Note that, traditional EM tasks find pairs of entities that are identical [53]. However, GEM might require finding out entity pairs satisfying a general binary relationship. Taking paper matching as an example, our goal is to find pairs between paper metadata and abstracts. Indeed, the relationship between them is whether they are relevant, which is more general beyond matching. Considering general binary relationship, we map the label y = yes into a set $\mathcal{V}_y = \{matched, similar, relevant\}$. Similarly, the label set for label y = no is $\mathcal{V}_y = \{mismatched, different, irrelevant\}$.

3.2 Training and Inference

A classification problem is transferred into a masked language modeling problem via prompt-tuning. In masked language modeling, we use confidence scores of all the words in $\mathcal{V}_y$ to construct the final score of the particular class y. Given a candidate pair x (which is mapped to $T(x)$) and its class y (which is mapped to $\mathcal{V}_y = \{w_1, \ldots, w_m\}$), the conditional probability is computed as:

$$P(y \mid x) = \frac{1}{m} \sum_{j}^{m} P\left([\text{MASK}] = w_j \mid T(x)\right) \qquad (1)$$

Training. The continuous prompt tokens can be parameterized by ϕ and optimized along with $\mathcal{M}$ during training. We tune the pre-trained model $\mathcal{M}$ (parameterized by θ) along with the additional prompt embeddings by using the cross-entropy loss function $\mathcal{L} = -\sum \log P(y \mid x; \theta, \phi)$. Here, we prompt-tune a pre-trained LM for the GEM task as follows:

(1) Design task-specific prompt templates and label words set.

(2) Initialize the network with parameters from the pre-trained LM and continuous prompt tokens.

(3) Train the network on the training set until convergence.

Step (1) is specifically designed for GEM tasks. Continuous prompt tokens in Step (2) are specifically designed to enable the model to find better prompts beyond $\mathcal{V}$ of $\mathcal{M}$ could express.

Inference. For inference, we aim to assign a label for the input, which can directly use Eq. 1 to predict the class of the current input instance based on predicted words of the [MASK] position.

4 LIGHTWEIGHT SELF-TRAINING

With prompt-tuning, we can stimulate the rich knowledge distributed in LMs, which achieves considerable performance under low-resource settings. To further improve the performance and

---

Algorithm 1: Lightweight Self-training

Input: the number **Iter** of iterations, a labeled train set D_L, an unlabeled train set D_U

Output: a student model $\mathcal{M}_{\theta,\phi}$

1 **for** $i \leftarrow 1$ **to Iter do**

2   Initialize a new teacher model $\mathcal{M}_{t,\theta,\phi,i}$

3   **for** $epoch \leftarrow 1$ **to** *Epochs of teacher* **do**

4   Train $\mathcal{M}_{t,\theta,\phi,i}$ using the train set D_L

5   ▷ Uncertainty-aware Pseudo-label Selection

6 $D_P \leftarrow$ Select pseudo-labels from D_U

7 $D_U \leftarrow D_U - D_P$

8 $D_L \leftarrow D_L \cup D_P$

9   Initialize a new student model $\mathcal{M}_{s,\theta,\phi,i}$

10   **for** $epoch \leftarrow 1$ **to** *Epochs of student* **do**

11   Train $\mathcal{M}_{s,\theta,\phi,i}$ using the train set D_L

12   ▷ Dynamic Data Pruning

13   **if** ($epoch$ mod *frequency of pruning*) = 0 **then**

14 $D_D \leftarrow$ Select useless samples from D_L

15 $D_L \leftarrow D_L - D_D$

16 **return** *the best student model* $\mathcal{M}_{\theta,\phi}$

---

avoid expensive self-training, we develop a generic lightweight self-training method.

4.1 Overview

Let $D_L = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N_L}$ and $D_U = \left\{x^{(i)}\right\}_{i=1}^{N_U}$ be a labeled dataset with N_L samples and an unlabeled dataset with N_U samples, respectively. Our lightweight self-training aims to boost the performance (i.e., effectiveness) using uncertainty meanwhile being more efficient and lightweight than traditional self-training via dynamic data pruning. We describe the lightweight self-training procedure, with its pseudo code presented in Algorithm 1. Given a labeled dataset D_L, a teacher model $\mathcal{M}_t$ is initialized and trained on D_L until convergence (Lines 2-4). Then the teacher model $\mathcal{M}_t$ produces pseudo-labels on D_U. After that, we introduce an uncertainty-aware pseudo-label selection strategy to select high-quality pseudo-labels D_P (Lines 5-6). Meanwhile, D_U and D_L are updated (Lines 7-8). Next, a student model $\mathcal{M}_s$ is initialized and trained on the updated D_L (Lines 9-11). To make self-training more lightweight and efficient, we present a dynamic data pruning strategy, which can prune useless samples and their labels in D_L every fixed number of epochs (Lines 12-15). Finally, we choose the best student model with the best performance on the validation set (Line 16). Noted that, the above steps can repeat iteratively. Since LST is general enough to incorporate with other approaches, it is possible to be widely used in practical low-resource applications.

4.2 Uncertainty-aware Pseudo-label Selection

Selecting high-quality pseudo-labels is a prerequisite for boosting self-training performance. Therefore, we aim at reducing the noise present in the selected samples to improve the overall performance. A straightforward way to select pseudo-labels is by selecting samples with high-confidence predictions. However, incorrect predictions can have high confidence scores in poorly calibrated networks [39]. Besides, if the teacher model already predicts some

samples with high confidence, then these is little to gain for student model with these samples [33]. Based on the observation that prediction uncertainties can be leveraged to negate the effect of poor calibration [39], we employ an uncertainty-aware pseudo-label selection strategy. Formally, uncertainty can be divided into *epistemic uncertainty* and *aleatoric uncertainty* [49]. The former comes from uncertainty in the parameters of the model, and the latter is uncertainty inherent in the data. We focus on quantifying *epistemic uncertainty*. Inspired by [33, 39], we use MC-Dropout [12] to obtain an uncertainty measure by calculating the standard deviation of a fixed number (e.g., 10 in our experiments) of stochastic forward passes. To avoid carefully chosen thresholds, we choose N_P samples with the least uncertainty after calculating the uncertainties of D_U:

$$D_P = \left\{ \left(x^{(i)}, \tilde{y}^{(i)} \right) \right\}_{i=1}^{N_P} = \text{Top-}N_P \left(D_U \mid -u^{(i)} \right) \quad (2)$$

Here, $N_P = N_U \cdot u_r$, $u^{(i)}$ is the uncertainty of the sample, $\tilde{y}^{(i)}$ is the pseudo-label produced by the teacher model, and u_r is the proportion of the unlabeled samples. The time complexity of the uncertainty estimation is $O(|N_P| \times log(|N_U|))$. Uncertainty-aware pseudo-label selection makes the self-training process more effective, which will be analyzed in Section 5.5.

4.3 Dynamic Data Pruning

Traditional self-training can be expensive as the training set grows, resulting in more training time. Recently, Paul et al. [36] show that the Error L2-Norm (EL2N) score can identify important examples early in training. In other words, it can prune significant fractions of useless training data without sacrificing test accuracy, which can reduce the number of the training set and training time. Inspired by EL2N [36] and MC-Dropout [12], we combine these approaches and propose MC-EL2N, which is able to quantify importance scores more stably. Formally, the MC-EL2N score of a training sample (x, y) is defined as $\frac{\sum_{i=1}^{n} ||\mathcal{M}(x) - y||_2}{n}$, where n is the number of stochastic forward passes. Similarly, to avoid carefully chosen thresholds, we choose N_D samples with the least MC-EL2N score after quantifying the importance of D_L:

$$D_D = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N_D} = \text{Top-}N_D \left(D_L \mid -e^{(i)} \right) \quad (3)$$

Here, $N_D = N_L \cdot e_r$, $e^{(i)}$ is the MC-EL2N score of the sample, and e_r is the proportion of the labeled samples. Similar to the uncertainty estimation, this process can be computed efficiently in $O(|N_D| \times log(|N_L|))$ time. We prune the useless samples every fixed epochs using dynamic data pruning, making the self-training process more lightweight and efficient. We will confirm the efficiency of dynamic data pruning, to be presented in Section 5.4.

5 EXPERIMENTS

In this section, we experimentally evaluate the proposed PromptEM on eight real-world datasets. We aim at answering the following research questions:
- **RQ1**: How does PromptEM perform compared with the state-of-the-art methods under low-resource settings?
- **RQ2**: How does each module affect the overall performance of the model?
- **RQ3**: How does PromptEM perform compared with state-of-the-art approaches in terms of efficiency?

Table 1: Statistics of the datasets used in our experiments. "All" denotes the total number of labeled samples, and "Train" represents the number of training samples used in our default low-resource setting.

Datasets	Domain	Left Table		Right Table		Labeled Examples		
		#row	#attr	#row	#attr	All	% rate	Train
REL-HETER	restaurant	534	6.00	332	7.00	567	10%	57
SEMI-HOMO	citation	2,616	8.65	64,263	7.34	17,223	5%	861
SEMI-HETER	book	22,133	12.28	23,264	12.03	1,240	10%	124
SEMI-REL	movie	29,180	8.00	32,823	13.81	1,309	10%	131
SEMI-TEXT-w	product	9,234	10.00	9,234	1.00	5,540	10%	554
SEMI-TEXT-c	product	20,897	10.00	20,897	1.00	12,538	5%	627
REL-TEXT	citation	2,616	1.00	2,295	6.00	7,417	10%	742
GEO-HETER	geo-spatial	2,469	5.00	2,788	4.00	2,500	10%	250

- **RQ4**: Why do we choose these key modules (i.e., continuous templates and uncertainty-aware pseudo-label selection)?

5.1 Experimental Setup

Dataset. We use all the seven real-world benchmark datasets with different structures from Machamp [53] and one geospatial dataset (GEO-HETER) [2]. The statistics of datasets are summarized in Table 1. Each dataset consists of the left and right tables of entities with possibly different formats (i.e., relational (REL) format, semi-structured (SEMI) format, or textual (TEXT) format). When they are of the same format, they can have a homogeneous (HOMO) or heterogeneous (HETER) schema. We use rate% of labeled data as training set, and use the same train/valid/test splits as Machamp.

Baselines. We compare PromptEM with seven SOTA EM methods, among which three (i.e., Ditto, DADER, and Rotom) have made efforts to *low-resource EM*, and TDmatch is an unsupervised matching method for structural and textual data. We implement baselines following the original papers and corresponding public codes.
- **DeepMatcher** [32] is an entity matching framework that uses RNN architecture to aggregate the attribute values and then align the aggregated representations of the attributes. We use the default setting of the hybrid model.
- **BERT** [6] is fine-tuned to treat GEM as a sequence pair classification task. Following [53], we tune the hyper-parameters for BERT-based baselines (i.e., BERT, SentenceBERT, Ditto, and Rotom) by doing a grid search and select the one with the best performance.
- **SentenceBERT** [38] proposes a siamese architecture for pre-trained LMs for sentence matching, and could also be applied to the task of GEM.
- **Ditto** [24] is the SOTA EM approach that fine-tunes a pre-trained LM with three optimizations (e.g., domain knowledge).
- **Rotom** [31] proposes a meta-learning framework that selects and weights the augmented data to better fine-tune the LMs.
- **DADER** [48] presents a transfer learning based EM framework via domain adaptation. For the source dataset, we use all the training samples. For the target dataset, we use the same low-resource training samples as other supervised methods.
- **TDmatch** [1] is an unsupervised approach to match textual and structured data using graph creation and random walk. All hyper-parameters are the same as the original paper.

Table 2: Results of all the methods under the default low-resource setting.

Methods	REL-HETER			SEMI-HOMO			SEMI-HETER			SEMI-REL			SEMI-TEXT-c			SEMI-TEXT-w			REL-TEXT			GEO-HETER		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
DeepMatcher	0.0	0.0	0.0	74.6	72.9	73.8	78.3	22.6	35.1	70.1	44.8	54.7	23.0	39.6	29.1	23.5	1.9	3.5	36.9	17.1	23.4	28.9	90.9	43.8
BERT	100	90.9	95.2	90.1	93.2	91.6	63.6	17.6	27.6	92	94.5	93.3	56.9	47.2	51.6	19.6	20.9	20.2	27.1	26.6	26.9	70.8	90.4	79.4
SentenceBERT	100	90.9	95.2	92.6	93.6	93.1	81.5	13.8	23.7	83.2	100	90.8	60.0	51.3	55.3	26.2	21.3	23.5	36.4	52.0	42.9	74.5	72.1	73.3
Ditto	100	86.4	92.7	90.2	90.3	90.2	79.3	14.5	24.5	88.0	88.5	88.3	56.8	47.1	51.5	29.5	31.3	30.3	34.7	50.5	41.1	74.7	87.2	80.5
DADER	81.8	81.8	81.8	81.5	91.4	86.2	98.4	37.7	54.6	87.6	96.2	91.7	15.0	87.4	25.6	11.4	100	20.5	26.1	64.6	37.2	54.2	92.7	68.4
Rotom	100	77.3	87.2	89.2	94.3	91.7	83.3	15.7	26.5	95.8	88.0	91.7	68.0	54.6	60.5	43.6	34.1	38.3	51.9	45.5	48.5	76.7	78.5	77.6
TDmatch	56.4	100	72.1	93.7	42.0	58.0	97.2	88.1	**92.4**	97.5	85.8	91.3	69.0	10.4	18.0	42.3	14.2	21.3	80.2	47.3	59.5	72.8	73.0	72.9
PromptEM	100	100	**100**	94.2	94.1	**94.2**	93.9	57.9	71.6	91.4	98.9	**95.0**	80.6	65.5	72.3	44.9	37.9	**41.1**	61.2	61.5	**61.4**	78.8	89.9	84.0
PromptEM w/o PT	100	95.5	97.7	90.5	94.8	93.3	39.5	52.2	45.0	83.8	53.6	65.3	56.3	55.3	55.8	31.5	19.0	23.7	22.1	55.9	31.7	79.3	85.1	82.1
PromptEM w/o LST	100	100	**100**	94.2	94.1	**94.2**	91.7	27.7	42.5	91.4	98.9	**95.0**	73.4	59.1	65.5	35.8	31.8	33.7	58.0	60.4	59.2	76.4	90.4	82.8
PromptEM w/o DDP	100	100	**100**	93.0	94.8	93.9	80.5	57.2	66.9	92.1	95.1	93.6	79.0	68.9	**73.6**	47.3	33.2	39.0	55.5	63.3	59.2	85.0	88.1	**86.5**

Table 3: Results of all the methods under the extremely challenging low-resource setting.

Methods	REL-HETER			SEMI-HOMO			SEMI-HETER			SEMI-REL			SEMI-TEXT-c			SEMI-TEXT-w			REL-TEXT			GEO-HETER		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
DeepMatcher	32.3	45.5	37.7	46.9	22.5	30.4	28.6	40.3	33.4	43.0	92.9	58.8	0.0	0.0	0.0	28.6	1.0	1.8	9.1	0.2	0.4	29.1	100	45.1
BERT	100	95.5	97.7	85.9	82.4	85.8	78.8	16.4	27.1	97.0	71.6	82.4	21.2	31.4	25.3	18.4	8.5	11.7	18.6	9.9	12.9	45.8	78.5	57.8
SentenceBERT	95.5	95.5	95.5	86.8	73.3	79.5	100	14.5	25.3	71.0	74.9	72.9	20.8	13.8	16.6	23.1	8.5	12.5	15.5	4.1	6.4	53.8	64.8	58.8
Ditto	100	81.8	90.0	80.9	81.8	81.4	95.2	12.6	22.2	78.7	95.1	86.1	13.9	100	24.3	12.6	69.7	21.4	18.0	99.3	30.4	33.0	85.4	47.6
DADER	88.9	72.7	80.0	75.9	86.6	80.9	47.1	65.0	54.6	94.0	86.3	90.0	41.7	0.9	1.7	12.4	7.1	9.0	60.2	11.9	19.9	63.0	87.9	73.4
Rotom	100	95.5	97.7	80.1	93.5	86.2	77.8	17.6	28.7	96.3	85.8	90.8	23.5	31.4	26.9	22.0	5.2	8.4	19.8	23.9	21.6	72.0	78.3	75.0
TDmatch	56.4	100	72.1	93.7	42.0	58.0	97.2	88.1	**92.4**	97.5	85.8	91.3	69.0	10.4	18.0	42.3	14.2	21.3	80.2	47.3	**59.5**	72.8	73.0	72.9
PromptEM	100	100	**100**	86.1	92.2	**89.0**	93.9	28.9	44.2	94.0	94.5	**94.3**	40.8	29.0	**33.9**	15.7	46.9	**23.6**	26.5	50.2	34.7	78.0	81.9	**79.9**

Implementation details of PromptEM. We implement PromptEM in PyTorch [35], the Transformers library [54] and the OpenPrompt library [8]. We use RoBERTa-base [27] as the backbone structure of our model in all the experiments. Unless particularly specified, the experimental results are conducted under the low-resource setting shown in Table 1. In all the experiments, the max sequence length is set to 512; the learning rate is set to 2e-5; the batch size is set to 32; the number of iterations for self-training is set to 1; and the number of passes for MC-Dropout is set to 10. We use AdamW as the optimizer for training, fix the epochs of training the teacher model to 20, and set the epochs of training the student model to 30. We prune the train set for every 8 epochs. We tune the hyper-parameters by doing a grid search and selecting the one with the best performance. Specifically, the continuous template is selected from $\{T_1(\cdot), T_2(\cdot)\}$, u_r is selected from $\{0.05, 0.10, 0.15, 0.20, 0.25\}$, and e_r is selected from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. We select the epoch with the highest F1-score on the validation set, and report the values of precision, recall, and F1-score on the test set. All the experiments are conducted on a machine with an Intel Xeon Silver 4216 CPU, an NVIDIA A100 GPU and 512GB memory. We use the same serializing method as PromptEM to implement each baseline method and report the performance under their optimal settings.

Evaluation metrics. Following related studies [15, 24, 48], we employ three widely-used classification metrics, namely, precision (P), recall (R), F1-score (F).

5.2 Main Results (RQ1)

Results under the default low-resource setting. We first verify the performance under low-resource setting of PromptEM using the above seven baselines. The benchmark results of all methods are reported in Table 2. DeepMatcher achieves the worst performance, since it does not leverage the recent advances in pre-trained LMs. Existing low-resource EM approaches (i.e., Ditto, DADER, and Rotom) achieve relatively poor performance, because the GEM problem is more intractable than EM (e.g., heterogeneous tables). In particular, TDmatch is not stable across different datasets due to the absence of label guidance, which can achieve the best F1-score on SEMI-HETER but only 18.0 F1-score on SEMI-TEXT-c. TDmatch outperform other LM-based approaches on SEMI-HETER. The reason is that SEMI-HETER has lots of numeric attributes, i.e., 53% attribute values are digits. It is well known that LMs are not good at understanding digits [50]. Also, we find that the scalability of TDmatch is extremely poor, which will be confirmed in Section 5.4.

Effectiveness to different low-resource settings. We reduce the training rate from 25% to 5% to see the performance under different low-resource settings. Experimental results are depicted in Figure 3. We observe that PromptEM achieves SOTA performance in most cases, while TDmatch and DADER achieve unstable results across different datasets due to lacking the guidance of labels and the heterogeneity of datasets. We also evaluate methods in a more challenging setting, i.e., the number of available data for training is only 80 for all the datasets. This setting is extremely challenging for supervised methods, e.g., only using 0.46% labeled examples on SEMI-HOMO. As shown in Table 3, PromptEM achieves SOTA performance on most datasets, which demonstrates the great robustness of PromptEM compared to baselines. Moreover, it shows the outstanding scalability of PromptEM, which achieves considerable performance only using a small number of labeled examples. Overall, our PromptEM is superior to all the baselines in almost all the cases under various low-resource settings. As mentioned in Challenge I, there is a significant gap between objective forms
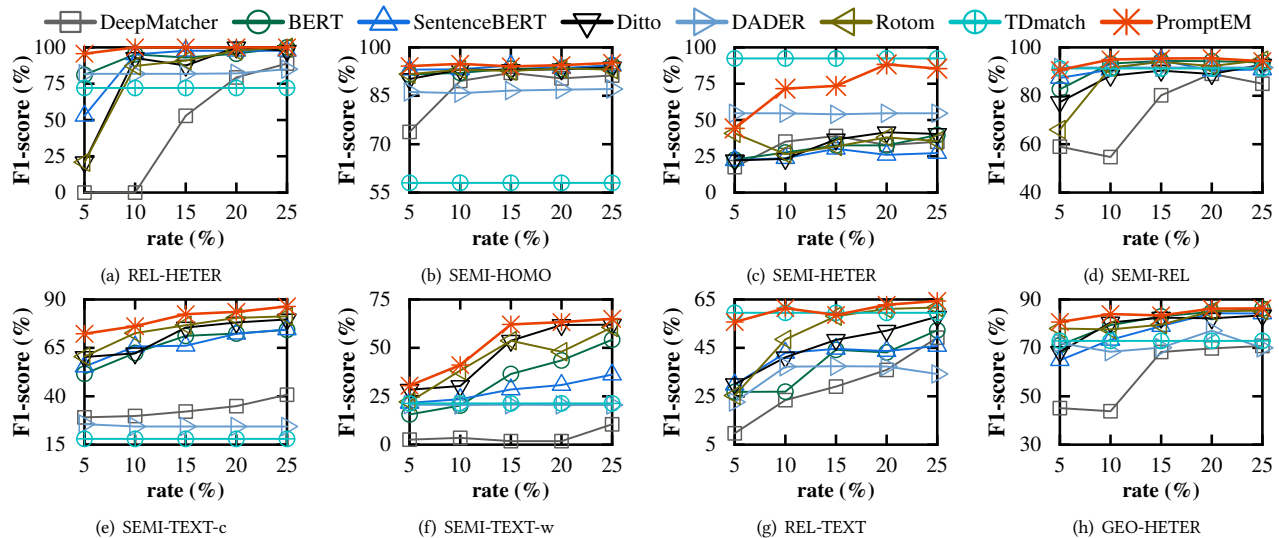
Figure 3: Comparison results under different low-resource settings (%).

Table 4: Efficiency comparison between PromptEM and its competitors, including the running time and memory usage. "s" denotes seconds, "m" denotes minutes, "h" denotes hours, and "G" represents gigabytes. Due to the limitation of space, we use the abbreviations of datasets.

Datasets	SBERT		Rotom		TDmatch		PromptEM-		PromptEM	
	T.	M.	T.	M.	T.	M.	T.	M.	T.	M.
R-H	28.8s	22.9G	35.4s	32.8G	14.0m	6.2G	38.3s	27.4G	26.6s	27.4G
S-HO	2.4m	29.5G	8.2m	32.8G	51.0h	41.4G	11.5m	29.2G	7.4m	29.2G
S-HE	43.4s	24.9G	1.8m	35.8G	102.8h	105.3G	1.6m	29.0G	1.5m	29.0G
S-R	50.1s	35.8G	2.6m	32.8G	120.3h	131.5G	1.4m	30.6G	1.1m	30.6G
S-T-c	2.0m	36.3G	20.2m	32.8G	10.7h	25.4G	20.8m	29.2G	11.5m	29.2G
S-T-w	1.8m	35.6G	11.2m	32.8G	2.2h	12.9G	6.1m	29.2G	5.3m	29.2G
R-T	2.3m	34.4G	11.1m	29.7G	5.6h	50.5G	8.1m	30.6G	5.6m	30.6G
G-H	49.2s	27.9G	1.3m	32.8G	36.2m	16.7G	6.2m	30.6G	4.6m	30.6G

in pre-training and fine-tuning. This gap hinders the transfer and adaptation of knowledge in LMs for GEM tasks, which restricts taking full advantage of knowledge in LMs. Prompt-tuning is able to bridge the gap of objective forms between pre-training and fine-tuning [18, 25]. Thus, we can stimulate the rich knowledge distributed in LMs through designing GEM-specific prompt-tuning [18, 25]. In summary, prompt-tuning has the potential to outperform fine-tuning for those tasks based on LMs.

5.3 Ablation Study (RQ2)

Next, we study the effectiveness of each module in PromptEM (i.e., prompt-tuning (PT), lightweight self-training (LST), dynamic data pruning (DDP)) by comparing PromptEM with its variants without the key module. The results are listed in Tables 2.

PromptEM vs. PromptEM w/o PT. PromptEM w/o PT denotes that we fine-tune the LM instead of prompt-tuning. It is observed that the use of prompt-tuning contributes to a large portion of the performance gain. The F1-score drops 15.7% on average under the low-resource setting. This confirms that prompt-tuning greatly helps to stimulate the rich knowledge distributed in the LM.

PromptEM vs. PromptEM w/o LST. We use LST to boost the performance under low-resource settings. We can observe that LST can bring performance improvement in most cases. For example, LST brings 6.8% improvement on SEMI-TEXT-c. Also notice that LST brings relatively low improvement on some datasets. This is attributes to the nature of the datasets, as it is relatively much easier for PromptEM to achieve the extremely high performance, e.g., 100% F1-score on REL-HETER.

PromptEM vs. PromptEM w/o DDP. We can observe that DDP can prune useless training data without sacrificing test accuracy. It is worth noting that DDP can prune training data while slightly improving test accuracy in some datasets. This is because DDP makes the model focus on important and useful training data.

5.4 Efficiency Analysis (RQ3)

We further explore the efficiency of our proposed PromptEM in terms of training time and memory usage, and the results are presented in Table 4. SBERT denotes SentenceBERT, and PromptEM-represents PromptEM without dynamic data pruning.

PromptEM vs. best baselines. Due to the limitation of space, we report PromptEM with the other evaluated approaches that achieve the best quality of GEM results in the corresponding categories, i.e., the normal EM method SBERT, the low-resource EM approach Rotom, and the unsupervised matching method TDmatch. As observed, PromptEM needs more training time than SBERT to obtain the SOTA results. This demonstrates a trade-off between the effectiveness and efficiency of the GEM problem. To sum up, it is significant that spending a relatively longer time in achieving better matching results. Rotom requires two-stage training, incurring a long training process. SBERT, Rotom, and PromptEM need similar memory usage since they are all based on LMs. We would like to emphasize that TDmatch needs too much training time and memory usage, especially on relatively large datasets (e.g., 120.3 hours and 131.5 Gigabytes on SEMI-REL), which is very costly in real-world applications.

Table 5: Results of pseudo-label selection strategies.

Datasets	Uncertainty		Confidence		Clustering	
	TPR	TNR	TPR	TNR	TPR	TNR
REL-HETER	1	1	0.250	0.864	0.250	0.881
SEMI-HOMO	1	0.998	0.197	0.803	0.193	0.815
SEMI-HETER	1	0.963	0.979	0.985	0.350	0
SEMI-REL	1	1	0.426	0.583	0.432	0.602
SEMI-TEXT-c	0.969	1	0.113	0.897	0.128	0.879
SEMI-TEXT-w	0.333	0.967	0.056	0.928	0.114	0.912
REL-TEXT	0.910	0.966	0.194	0.820	0.148	0.846
GEO-HETER	0.867	1	0.644	0.758	0.236	0.738

PromptEM vs. PromptEM-. We also compare PromptEM with PromptEM- to evaluate the efficiency of dynamic data pruning. It is observed that DDP greatly helps to reduce the training time, i.e., reduce 26.1% time on average. This is because the proposed MC-EL2N is able to quantify useless training data effectively. Meanwhile, DDP does not bring extra memory usage as it does not require any new model parameters. As analyzed in Section 5.3, DDP does not hurt the performance. This further demonstrates that PromptEM is effective and efficient in solving the GEM problem.

5.5 Model Variants (RQ4)

Finally, we investigate the performance of PromptEM using alternative modules by conducting the following experiments.

Effect of template choices. We verify the effect of different templates, i.e., continuous $T_1(\cdot)$, hard-encoding $T_1(\cdot)$, continuous $T_2(\cdot)$ and hard-encoding $T_2(\cdot)$. Their average F1-scores on all datasets are 74.4, 67.8, 77.0, and 74.5, respectively. Continuous templates achieve better performance than hard-encoding templates. This further validates the effectiveness of the proposed continuous templates, which can find better continuous prompts beyond the original vocabulary $\mathcal{V}$ of $\mathcal{M}$ could express.

Effect of label words choices. We compare our designed label words with a simple one (i.e., matched and mismatched). Using continuous T_1 and T_2, our designed label words achieve +5.2% and +9.4% average F1-score improvements over the simple one, respectively. This confirms the effectiveness of our designed label words. In other words, considering the more general relationship between entities is beneficial to the predictions.

Pseudo-label selection strategies. We consider several pseudo-label selection strategies, including uncertainty [39], confidence, and clustering [9]. We fix u_r to 0.1 on all datasets. Similarly, confidence and clustering both select the samples whose scores are in the top 10%. Following [15], we use true-positive rate (TPR) and true-negative rate (TNR) to evaluate the quality of the pseudo-labels generated by different strategies. The results are reported in Table 5. As expected, uncertainty can achieve state-of-the-art performance when generating pseudo-labels. It confirms the effectiveness of the uncertainty-aware pseudo-label selection strategy.

6 RELATED WORK

6.1 Entity Matching

Entity Matching (EM) is one of the fundamental and significant tasks in data management. Many efforts have devoted to develop effective approaches for EM, including rule-based methods [11, 43, 52], crowdsourcing-based methods [16, 30, 51], and traditional ML-based methods [3, 5, 23, 40]. Recently, deep learning has been used widely in EM [10, 32], and achieved promising results. However, lots of labeled examples are still needed for those DL-based approaches, which is extremely expensive in practice. To decrease the demand for high-quality training data, Ditto [24] and Rotom [31] perform well with the help of LMs and data augmentation (DA) techniques. Some other attempts have also been made to enhance the performance via information fusion [13, 14, 56], active learning [22, 34], and transfer learning [21, 28, 46, 48, 57]. Nonetheless, these methods only focus on EM tasks in low-resource scenarios but perform poorly on GEM tasks. TDmatch [1] first attempts to match textual content and structured data under an unsupervised setting. However, it is not scalable on large-scale datasets, which makes it hard to be used in practical.

6.2 Prompt-tuning

Despite the success of fine-tuning pre-trained LMs [6, 27], the huge objective form gap between pre-training and fine-tuning still hinders the full use of pre-trained knowledge for downstream tasks [25, 26]. The birth of GPT-3 [4] is the seminal work that stimulates the development of prompt-tuning, which applies hand-encoding prompts for tuning and achieves impressive performance on various tasks, especially under the low-resource settings. Following GPT-3, many hand-encoding prompts [7, 26], continuous prompts [19, 26], and automatic prompt search [42] are widely explored to enhance the expressiveness of the prompt. The burst of prompt-tuning has led to significant advancement in many areas such as entity typing [7], machine translation [44], and image classification [37]. However, for the first time, we introduce prompt-tuning in EM for the better usage of pre-trained LMs. PromptEM provides a good connection to recent NLP advancements with applications to the data management task.

7 CONCLUSIONS

In this paper, we study the problem of low-resource generalized entity matching via our presented PromptEM. For the first time, PromptEM introduces prompt-tuning to cast GEM as a cloze-style task to bridge the gap between pre-training and fine-tuning. It is non-trivial as prompt templates are not generic and need to be specially designed for the task. To this end, we design GEM-specific templates and label words set for the GEM problem. To further improve the performance in low-resource settings, we develop a generic lightweight self-training method using uncertainty, which is a nice combination and can be efficient and lightweight by dynamic data pruning. Extensive experimental results on eight real-world datasets with different structures demonstrate the superiority of PromptEM compared with the state-of-the-art approaches. In the future, we plan to explore a general prompt-tuning method to support more data management tasks (e.g., data cleaning).

REFERENCES

[1] Naser Ahmadi, Hansjorg Sand, and Paolo Papotti. 2022. Unsupervised Matching of Data and Text. In *ICDE*. 1058–1070.

[2] Pasquale Balsebre, Dezhong Yao, Gao Cong, and Zhen Hai. 2022. Geospatial Entity Resolution. In *WWW*. 3061–3070.

[3] Mikhail Bilenko and Raymond J Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*. 39–48.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*. 1877–1901.

[5] William W Cohen and Jacob Richman. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *SIGKDD*. 475–480.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Ning Ding, Yulin Chen, Xu Han, Guangwei Xu, Pengjun Xie, Hai-Tao Zheng, Zhiyuan Liu, Juanzi Li, and Hong-Gee Kim. 2021. Prompt-learning for fine-grained entity typing. *arXiv preprint arXiv:2108.10604* (2021).

[8] Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun. 2021. Openprompt: An open-source framework for prompt-learning. *arXiv preprint arXiv:2111.01998* (2021).

[9] Thomas Dopierre, Christophe Gravier, Julien Subercaze, and Wilfried Logerais. 2020. Few-shot pseudo-labeling for intent detection. In *COLING*. 4993–5003.

[10] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *PVLDB* 11, 11 (2018), 1454–1467.

[11] Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2014. NADEEF/ER: Generic and interactive entity resolution. In *SIGMOD*. 1071–1074.

[12] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*. 1050–1059.

[13] Yunjun Gao, Xiaoze Liu, Junyang Wu, Tianyi Li, Pengfei Wang, and Lu Chen. 2022. ClusterEA: Scalable Entity Alignment with Stochastic Training and Normalized Mini-batch Similarities. In *KDD*. 421—431.

[14] Congcong Ge, Xiaoze Liu, Lu Chen, Baihua Zheng, and Yunjun Gao. 2022. LargeEA: Aligning Entities for Large-scale Knowledge Graphs. *PVLDB* 15, 2 (2022), 237–245.

[15] Congcong Ge, Pengfei Wang, Lu Chen, Xiaoze Liu, Baihua Zheng, and Yunjun Gao. 2021. CollaborEM: A Self-supervised Entity Matching Framework Using Multi-features Collaboration. *TKDE* (2021).

[16] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*. 601–612.

[17] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *ICASSP*. 6645–6649.

[18] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.

[19] Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. Ptr: Prompt tuning with rules for text classification. *arXiv preprint arXiv:2105.11259* (2021).

[20] Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2019. Revisiting Self-Training for Neural Sequence Generation. In *ICLR*.

[21] Di Jin, Bunyamin Sisman, Hao Wei, Xin Luna Dong, and Danai Koutra. 2021. Deep transfer learning for multi-source entity linkage via domain adaptation. *PVLDB* 15, 3 (2021), 465–477.

[22] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource Deep Entity Resolution with Transfer and Active Learning. In *ACL*. 5851–5861.

[23] Pradap Konda, Sanjib Das, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, et al. 2016. Magellan: toward building entity matching management systems over data science stacks. *PVLDB* 9, 13 (2016), 1581–1584.

[24] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *PVLDB* 14, 1 (2020), 50–60.

[25] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586* (2021).

[26] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *arXiv preprint arXiv:2103.10385* (2021).

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[28] Michael Loster, Ioannis Koumarelas, and Felix Naumann. 2021. Knowledge transfer for entity resolution with siamese neural networks. *JDIQ* 13, 1 (2021), 1–25.

[29] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. *PVLDB* 1, 2001 (2001), 49–58.

[30] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. 2011. Human-powered sorts and joins. *arXiv preprint arXiv:1109.6881* (2011).

[31] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond. In *SIGMOD*. 1303–1316.

[32] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*. 19–34.

[33] Subhabrata Mukherjee and Ahmed Awadallah. 2020. Uncertainty-aware self-training for few-shot text classification. *NeurIPS*, 21199–21212.

[34] Youcef Nafa, Qun Chen, Zhaoqiang Chen, Xingyu Lu, Haiyang He, Tianyi Duan, and Zhanhuai Li. 2022. Active deep learning on entity resolution by risk sampling. *Knowledge-Based Systems* 236 (2022), 107729.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: an imperative style, high-performance deep learning library. In *NeurIPS*. 8026–8037.

[36] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep Learning on a Data Diet: Finding Important Examples Early in Training. *NeurIPS* 34.

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *ICML*. 8748–8763.

[38] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*.

[39] Mamshad Nayeem Rizve, Kevin Duarte, Yogesh S Rawat, and Mubarak Shah. 2021. In defense of pseudo-labeling: An uncertainty-aware pseudo-label selection framework for semi-supervised learning. In *ICLR*.

[40] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *SIGKDD*. 269–278.

[41] Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. 2020. A mathematical exploration of why language models help solve downstream tasks. *arXiv preprint arXiv:2010.03648* (2020).

[42] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *EMNLP*. 4222–4235.

[43] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing entity matching rules by examples. *PVLDB* 11, 2 (2017), 189–202.

[44] Zhixing Tan, Xiangwen Zhang, Shuo Wang, and Yang Liu. 2022. MSP: Multi-Stage Prompting for Making Pre-trained Language Models Better Translators. In *ACL*. 6131–6142.

[45] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: a design space exploration. *PVLDB* 14, 11 (2021), 2459–2472.

[46] Saravanan Thirumuruganathan, Shameem A Puthiya Parambath, Mourad Ouzzani, Nan Tang, and Shafiq Joty. 2018. Reuse and Adaptation for Entity Resolution through Transfer Learning. *arXiv e-prints* (2018), arXiv–1809.

[47] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. 613–622.

[48] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Chengliang Chai, Guoliang Li, Ruixue Fan, and Xiaoyong Du. 2022. Domain Adaptation for Deep Entity Resolution. In *SIGMOD*. 443–457.

[49] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. 2020. Uncertainty estimation using a single deep deterministic neural network. In *ICML*. 9690–9700.

[50] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP Models Know Numbers? Probing Numeracy in Embeddings. In *EMNLP*. 5307–5315.

[51] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *PVLDB* 5, 11 (2012).

[52] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. 2011. Entity matching: How similar is similar. *PVLDB* 4, 10 (2011), 622–633.

[53] Jin Wang, Yuliang Li, and Wataru Hirota. 2021. Machamp: A Generalized Entity Matching Benchmark. In *CIKM*. 4633–4642.

[54] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[55] Qiantong Xu, Alexei Baevski, Tatiana Likhomanenko, Paden Tomasello, Alexis Conneau, Ronan Collobert, Gabriel Synnaeve, and Michael Auli. 2021. Self-training and pre-training are complementary for speech recognition. In *ICASSP*. 3030–3034.

[56] Zijun Yao, Chengjiang Li, Tiansi Dong, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Yichi Zhang, and Zelin Dai. 2021. Interpretable and Low-Resource Entity Matching via Decoupling Feature Learning from Decision Making. In *ACL*. 2770–2781.

[57] Chen Zhao and Yeye He. 2019. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *WWW*. 2413–2424.