Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration

JIANHONG TU, Renmin University of China, China
JU FAN*, Renmin University of China, China
NAN TANG, QCRI / HKUST (GZ), Qatar / China
PENG WANG, Renmin University of China, China
GUOLIANG LI, Tsinghua University, China
XIAOYONG DU, Renmin University of China, China
XIAOFENG JIA, Beijing Big Data Centre, China
SONG GAO, Beijing Big Data Centre, China

Data matching – which decides whether two data elements (*e.g.,* string, tuple, column, or knowledge graph entity) are the "same" (*a.k.a.* a match) – is a key concept in data integration, such as entity matching and schema matching. The widely used practice is to build task-specific or even dataset-specific solutions, which are hard to generalize and disable the opportunities of knowledge sharing that can be learned from different datasets and multiple tasks.

In this paper, we propose **Unicorn**, a unified model for generally supporting common data matching tasks. Moreover, this unified model can enable knowledge sharing by learning from multiple tasks and multiple datasets, and can also support zero-shot prediction for new tasks with zero labeled matching/non-matching pairs. However, building such a unified model is challenging due to heterogeneous formats of input data elements (*e.g.,* strings, tuples, columns, trees, graphs, and so on) and various matching semantics of multiple tasks. To address the challenges, **Unicorn** employs one generic `Encoder` that converts any pair of data elements (a, b) into a learned representation, and uses a `Matcher`, which is a binary classifier, to decide whether a matches b. To align matching semantics of multiple tasks, **Unicorn** also adopts a mixture-of-experts model that enhances the learned representation into a better representation, which can further boost the performance of predictions. We conduct extensive experiments on 20 datasets of seven well-studied data matching tasks, including entity matching, entity linking, entity alignment, column type annotation, string matching, schema matching, and ontology matching, and find that our unified model can achieve better performance on most tasks and on average, compared with the state-of-the-art specific models trained for ad-hoc tasks and datasets separately. Moreover, **Unicorn** can also well serve new matching tasks with zero-shot learning.

CCS Concepts: • **Information systems → Mediators and data integration**.

Additional Key Words and Phrases: data matching, data integration, multi-task learning

*Ju Fan is the corresponding author.

Authors' addresses: Jianhong Tu, Renmin University of China, Beijing, China, tujh@ruc.edu.cn; Ju Fan, Renmin University of China, Beijing, China, fanj@ruc.edu.cn; Nan Tang, QCRI / HKUST (GZ), Doha / Guangzhou, Qatar / China, ntang@hbku.edu.qa; Peng Wang, Renmin University of China, Beijing, China, lisa_wang@ruc.edu.cn; Guoliang Li, Tsinghua University, Beijing, China, liguoliang@tsinghua.edu.cn; Xiaoyong Du, Renmin University of China, Beijing, China, duyong@ruc.edu.cn; Xiaofeng Jia, Beijing Big Data Centre, Beijing, China, jiaxf@jxj.beijing.gov.cn; Song Gao, Beijing Big Data Centre, Beijing, China, gaos@jxj.beijing.gov.cn.

2836-6573/2023/5-ART84 $15.00
https://doi.org/10.1145/3588938

ACM Reference Format:
Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration. *Proc. ACM Manag. Data* 1, 1, Article 84 (May 2023), 26 pages. https://doi.org/10.1145/3588938

1 INTRODUCTION

Data matching generally refers to the process of deciding whether two data elements are the "same" (*a.k.a.* a match) or not, where each data element could be of different classes such as string, tuple, column, and so on. Data matching is a key concept in data integration [13] and data preparation [9] that includes a wide spectrum of tasks. In this paper, we consider seven common data matching tasks, namely entity matching, entity linking, entity alignment, string matching, column type annotation, schema matching, and ontology matching, as shown in Figure 1.

Due to their importance, almost all aforementioned matching tasks have been studied for decades, and still remain to be important research topics. With the tremendous successes of deep learning, deep neural networks have been widely used for tackling various matching tasks, such as DeepMatcher [34] and Ditto [30] for entity matching (see *e.g.,* tutorial [4]), and TURL [10] and HNN [5] for column type annotation. Nevertheless, current deep learning based solutions for these matching tasks are task-specific or even dataset-specific, which are referred to as **specific models** in this paper.

There are two main **limitations of specific models**. First, they can only learn knowledge from specific tasks or datasets. That is, the learned knowledge cannot be shared across different models. For example, the knowledge learned by a column type annotation model (*e.g.,* TURL [10]) cannot be shared with an entity matching model (*e.g.,* Ditto [30]) due to different neural network designs, although very likely the two models can help each other. Second, one model has to be trained (or fine-tuned) only on the labeled examples of each task or dataset, which is inefficient and has a high monetary cost. For example, DeepMatcher [34] and Ditto [30] have to be fine-tuned on a new entity matching dataset with at least hundreds of labeled matching/non-matching pairs.

In this paper, we propose **`Unicorn`**, a unified model to support multiple data matching tasks, as shown in Figure 1. Compared with aforementioned specific models, **`Unicorn`** has the following notable advantages.

(1) ***Task unification*** that generalizes task-specific solutions into one unified model, thus achieving lower maintenance complexity and smaller model sizes, compared with specific models.
(2) ***Multi-task learning*** that enables the unified model to learn from multiple tasks and multiple datasets to make full use of *knowledge sharing*, which even outperforms specific models trained only on their own datasets separately.
(3) ***Zero-shot prediction*** that allows the model to make predictions for a new task or a new dataset with *zero* labeled matching/non-matching pairs.

Although several unified models have been recently studied in the NLP and CV communities [3, 41, 42], building such a unified model for supporting multiple data matching tasks is challenging.

(1) Data elements in the matching tasks can take ***heterogeneous formats***, such as tuples and columns in tabular data, entities in knowledge graphs, and plain text, which will increase the difficulty of task unification.
(2) Each data matching task may have its ***unique matching semantics***. For example, entity matching that matches two tuples is different from schema matching that matches two columns. Thus, it is non-trivial to enable knowledge sharing among multiple tasks.

To address the challenges, we develop a general framework **`Unicorn`**, which consists of three key modules: an `Encoder`, a `Mixture-of-Experts` layer, and a `Matcher` (see Figure 1). First, the `Encoder`

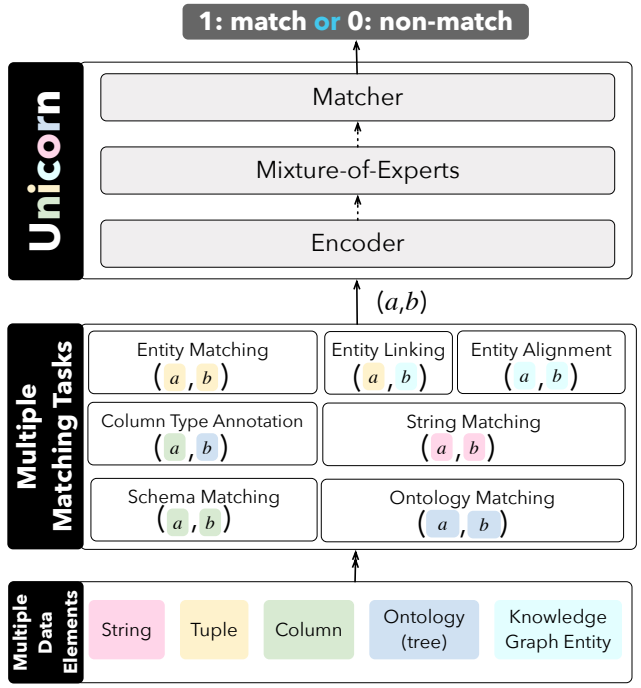


Fig. 1. **Unicorn** is a unified model for "data matching" task in data integration. So far, it supports seven matching tasks in data integration for a pair (a, b) where a or b may be of the same or different types of data elements. **Unicorn** consists of a unified `Encoder`, a `Mixture-of-Experts` layer, and a `Matcher` that will decide whether (a, b) is a match (1) or a non-match (0).

converts any pair of elements (a, b) with heterogeneous formats into a learned representation. To this end, we propose to serialize any pair of elements into text while still preserving their inherent structure, and employ a unified pre-trained language model for effective encoding. Second, to align matching semantics of various tasks or datasets, the `Mixture-of-Experts` module enhances the learned representation into a better representation by combining the knowledge from multiple "experts" controlled by a learned gating network. Third, the `Matcher` is a typical binary classifier, *e.g.,* a multilayer perceptron (MLP), to predict either 1 (for match) or 0 (for non-match) by taking the above representation as input.

Our notable contributions are summarized as follows.

(1) As far as we know, **Unicorn** is the first unified multi-tasking model towards supporting various matching tasks in data integration. We formally define the problem of data matching tasks and introduce an overview of the **Unicorn** framework (Section 2).
(2) We develop effective techniques for two key modules in the **Unicorn** framework. We introduce a unified representation learning method for the `Encoder` module (Section 3) and design effective methods for the `Mixture-of-Experts` layer (Section 4).
(3) We conduct a thorough evaluation on 20 datasets for the seven common data matching tasks shown in Figure 1. Extensive experiments show that **Unicorn**, as a unified model, outperforms the state-of-the-art specific models on most tasks and on average (Section 5 **Exp-3**). In addition, **Unicorn** can well serve new matching tasks with zero-shot learning (Section 5 **Exp-4**).

(4) We make a unified benchmark for multiple data matching tasks available at Github[1] . We standardize the format of datasets from different data matching tasks, making these datasets more convenient to use. We publish the source code of **Unicorn**, and provide the trained **Unicorn** model with multi-task training in Hugging Face[2], so that researchers and practitioners can either use it out-of-the-box, or fine-tune it for specific tasks.

2 PROBLEM AND SOLUTION OVERVIEW

This section formalizes the *data matching* problem (Section 2.1), and presents an overview of our **Unicorn** framework (Section 2.2).

2.1 Problem: Data Matching Tasks

Data Elements. In this paper, we consider a *data element* as a basic unit of information in data integration, which has a unique meaning. We consider data elements in the following five categories:

- **String.** A *string* is a sequence of words, which could be a noun or a natural language sentence, denoted as $\langle \mathtt{word}_i \rangle_{1 \le i \le k}$.
- **Tuple.** A *tuple* is a row contained in a table, consisting of a set of attribute-value pairs $\{(\mathtt{attr}_i, \mathtt{val}_i)\}_{1 \le i \le k}$, where $\mathtt{attr}_i$ and $\mathtt{val}_i$ are respectively the i-th attribute name and value of the tuple.
- **Column.** A *column* is a set of values $\{\mathtt{val}_i\}_{1 \le i \le k}$ of a particular attribute $\mathtt{attr}$ within a table, one value for each row of the table.
- **Ontology.** An *ontology* identifies and distinguishes hierarchical concepts and the relationships among the concepts. An ontology is formalized as a *tree* structure $\mathsf{ont} = \{\mathsf{node}_i, \mathsf{pnode}_i\}_{1 \le i \le k}$, where pnode_i is the parent of node_i. The unique node, which does not have a parent node, is called the *root* node.
- **Knowledge Graph Entity.** A *knowledge graph entity* (or *KG-entity* for short) describes a real-world entity, such as people, places, and things, in a knowledge graph. Formally, a knowledge graph (KG) is defined as a *graph* structure $\mathsf{KG} = (E, R, A, V, T_r, P_a)$, where $\mathtt{ent} \in E$, $\mathtt{rel} \in R$, $\mathtt{attr} \in A$, and $\mathtt{val} \in V$ represent an entity, a relation, an attribute, and an attribute value respectively. Moreover, $(\mathtt{ent}_i, \mathtt{rel}_k, \mathtt{ent}_j) \in T_r(\mathtt{ent}_i)$ denotes a relational triple, and $(\mathtt{attr}_k, \mathtt{val}_j) \in P_a(\mathtt{ent}_i)$ denotes an attribute-value pair.

Figure 2 shows example data elements with the categories of String (in pink color), Tuple (in yellow color), Column (in green color), Ontology (in blue color), and KG-Entity (in cyan color).

Data Matching. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets of data elements. The problem of *data matching* is to find all the pairs $(a_i, b_j) \in A \times B$ that are *matched*, where the semantic of whether (a_i, b_j) is matched or unmatched depends on the specific data matching tasks, as follows.

Data Matching Task Types. Based on various combinations of types of input pair (a, b) and the matching semantics, we consider the following seven common types of data matching tasks, as shown in Figure 2 For each task, we describe the existing solutions and our formal definitions in this paper.

(1) Entity matching refers to the task of determining whether two different tuples from two tables refer to the same real-world object [7, 18]. Existing solutions either define similarity functions based on aligned attributes (*e.g.,* Magellan [14]), use word embeddings (*e.g.,* DeepER [19] and DeepMatcher [34]), or piggyback pre-trained language models (*e.g.,* Ditto [30] and DADER [48]).

[1]https://github.com/ruc-datalab/Unicorn
[2]https://huggingface.co/RUC-DataLab/unicorn-plus-v1

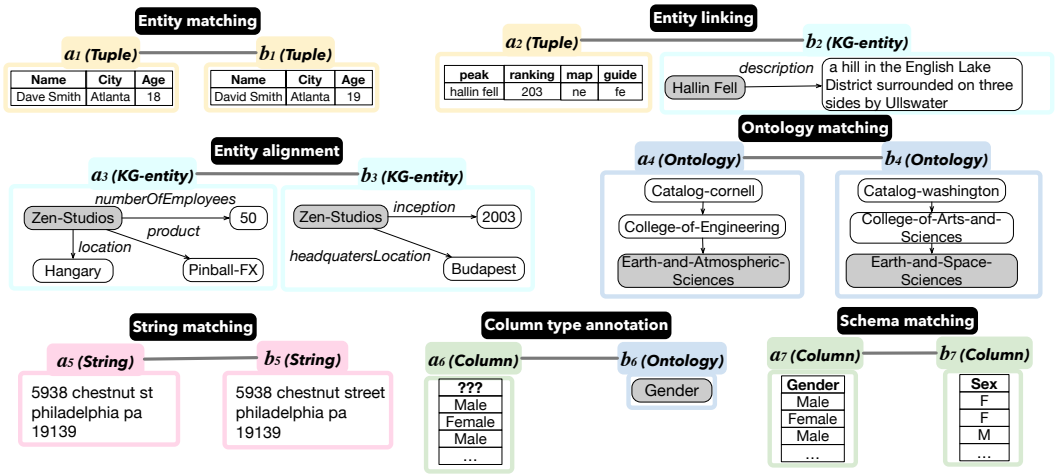


Fig. 2. Samples of common data matching tasks over data elements with the following five categories: String (in pink color), Tuple (in yellow color), Column (in green color), Ontology (in blue color), and KG-Entity (in cyan color).

We formalize entity matching as determining whether a (`Tuple`, `Tuple`) pair matches or not. For example, an entity matching task in Figure 2 determines whether the pair (a_1, b_1) of two records (with attributes `Name`, `City` and `Age`) refer to the same person. Note that, for some entity matching benchmarks, an entity could be of the JSON format (*e.g.,* the Web Data Commons dataset [35]). In this case, a JSON-entity can be first converted into a tuple-entity, such that it falls into our entity matching task.

(2) Entity linking refers to the task of determining whether a mention in a table refers to the same object with a `KG-Entity` in a knowledge base. A mention is represented by a tuple that contains multiple attributes. Existing solutions typically retrieve potential entities from knowledge bases and then sort them by calculating their similarities (*e.g.,* Hybrid II [20] and TURL [10]).

We formalize entity linking as determining whether a (`Tuple`, `KG-Entity`) pair matches or not. An example of entity linking is shown in Figure 2 to determine whether a tuple a_2 in a web table is matched with the "*Hallin Fell*" entity in a knowledge base.

(3) Entity alignment refers to a task of determining whether two KG-entities, typically from different knowledge graphs (*e.g.,* DBPedia and YAGO), are the same real-world object (see [56] for a survey). Existing solutions mainly learn entity embeddings and realize the matching of embeddings through graph neural networks (*e.g.,* GCN [26], CUEA [57]) or pre-trained language models (*e.g.,* BERT-INT [46]).

Entity alignment can be naturally formalized as determining whether a (`KG-Entity`, `KG-Entity`) pair matches or not. Typically, the two KG-entities to be matched, such as (a_3, b_3) in Figure 2, may have different attributes or relational triples, where a_3 has one attribute-value pair "*(numberOfEmployees, 50)*" and two relational triples "*(Zen-Studios, location, Hangary)*" and "*(Zen-Studios, product, Pinball-FX)*", and b_3 has one another attribute-value pair and one another relational triple.

(4) String matching refers to the task of determining whether two strings from two data sources are semantically the same or not. Existing solutions use string similarity functions [50] or machine learning methods such as the decision tree model to predict the results (*e.g.,* Smurf [39] and Falcon [8]).

We formalize the task as determining whether a (`String`, `String`) pair matches, *e.g.,* determining whether a_5 ("*5938 Chestnet St., Philadelphia, PA 19139*") and b_5 ("*5938 Chestnet Street, PHL, PA 19139*") in Figure 2 indicate the same address.

(5) Column type annotation typically determines the semantic types of a column in a table. Existing solutions use a single cell embedding or column embedding to represent a column through neural networks and then determine its type by similarity function or machine learning, where the type is a category (*e.g.,* HNN+P2Vec [5] and TURL [10]).

We formalize it as determining whether a (`Column`, `Ontology`) pair matches or not. Consider column a_6 in Figure 2, the problem is to decide whether an ontology b_6 (*e.g.,* "*Gender*") is an appropriate column type for the given Column a_6.

(6) Schema matching determines the correspondences between columns of two schemata from different tables. Existing solutions commonly define heuristic rules or calculate similarities between schemas (*e.g.,* COMA [12]).

We formalize the schema matching task as determining whether a (`Column`, `Column`) pair matches or not. Consider the pair of a_7 (named "*Gender*") and b_7 (named "*Sex*") in Figure 2: schema matching is the process of identifying whether there is semantic correspondence between these two columns, *e.g.,* referring to the same attribute of a person.

(7) Ontology matching finds correspondences between semantically related entities from different ontologies. Existing solutions calculate similarities such as *Jaccard* similarity cross different nodes of ontology and use machine learning models to predict the results (*e.g.,* GLUE [15]).

We formalize ontology matching as determining a (`Ontology`, `Ontology`) pair matches or not. Consider the pair (a_4, b_4) in Figure 2: ontology matching is to determine whether "*Earth-and-Atmospheric-Sciences*" from "*College-of-Engineering*" in "*Cornell*" and "*Earth-and-Space-Sciences*" from "*College-of-Arts-and-Sciences*" in "*Washington*" refer to the same specialized subject.

2.2 A Unified Multi-Tasking Framework

Given multiple data matching tasks, the widely used practice is to build task-specific solutions, *e.g.,* using different models or with various parameters. On the contrary, we introduce a unified multi-tasking framework called **`Unicorn`** for multiple data matching tasks, which has the following notable advantages.

- **Task unification** standardizes task-specific solutions into a unified framework, achieving lower development complexity, smaller model sizes and easier adoption of new tasks.
- **Multi-task learning** enables the possible opportunities of **knowledge sharing** among different data matching tasks compared with training each task separately.

Figure 3 shows an overview of the **`Unicorn`** framework. The basic idea is to unify multiple data matching tasks into a *text-to-prob* format due to its flexibility and extensibility, thus achieving a unified input and output of different tasks. Specifically, **`Unicorn`** unifies the input by serializing any pair (a, b) of data elements into a *text sequence*, and outputs a *probability* $\hat{y}$ to indicate if a and b match. To this end, **`Unicorn`** utilizes three main modules, namely `Encoder`, `Mixture-of-Experts` and `Matcher`. Next, we introduce the input and the main modules of **`Unicorn`** as follows.

Input: Multiple Data Matching Tasks. Instead of considering an individual data matching task, **`Unicorn`** takes as input a collection of data matching tasks, denoted as $\mathcal{T} = \{T_i\}$. In particular, each task $T_i = (A_i, B_i, \tau_i, \mathcal{D}_i)$ is composed of two sets of data elements to be matched, *i.e.,* A_i and B_i, and τ_i is the type of task T_i (see Section 2.1 for the supported task types). $\mathcal{D}_i \subset A_i \times B_i \times \{0, 1\}$ is a set of labeled examples, each of which denotes whether a pair of elements $a \in A_i$ and $b \in B_i$ is a match (*i.e.,* label 1) or a non-match (*i.e.,* label 0). Figure 3 (a) shows three example data matching tasks, *i.e.,*

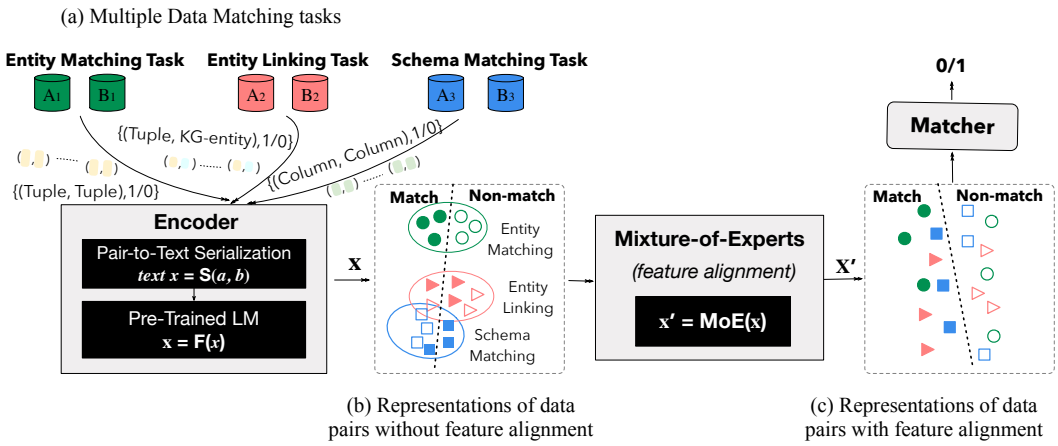


Fig. 3. An overview of our proposed **Unicorn** framework. (a) **Unicorn** learns from multiple data matching tasks or datasets. (b) A generic Encoder converts any pair of data elements (a, b) into a representation in the form of a high-dimensional vector. (c) A Mixture-of-Experts aligns matching semantics of multiple tasks by enhancing the learned representation into a better representation. Based on the representation, a Matcher, *i.e.,* a binary classifier, decides whether a matches b.

entity matching over A_1 and B_1, *entity linking* over A_2 and B_2 and *schema matching* over A_3 and B_3. Note that **Unicorn** is extensible that new task types can be easily supported.

Encoder: The Input Layer. Given an element pair (a, b) from any data matching task (*e.g.,* Figure 3 (a)), the aim of Encoder is to first serialize the pair into a *text sequence* x, and then map x into a high-dimensional vector-based representation $\mathbf{x}$, *i.e.,*

$$\mathbf{x} = \mathsf{F}(x) = \mathsf{F}(\mathsf{S}(a, b)), \tag{1}$$

where $\mathsf{S}(\cdot)$ is a generic function for serializing any data element pair (a, b) from the matching tasks in $\mathcal{T}$ into a text sequence, and $\mathsf{F}(\cdot)$ is a pre-trained language model (PLM) for deriving high-dimensional feature vectors from the serialized sequences. For example, in Figure 3 (b), circles, triangles and squares respectively represent feature vectors of the entity matching task, the entity linking task and the schema matching task, where a solid shape (resp. a hollowed shape) denotes a match (resp. a non-match).

Mixture-of-Experts: The Intermediate Layer. Although all the pairs from different tasks are mapped into one feature space, the distributions of their representations may not be aligned, as shown in Figure 3 (b). Consequently, it is hard to train a good Matcher.

To address the problem, we introduce an intermediate layer Mixture-of-Experts (MoE) [28, 33, 40, 44] to align the representations of different tasks, such that a good Matcher is easier to learn, as shown in Figure 3 (c). The basic idea is to transform original features of the pairs into an aligned feature space, *i.e.,* $\mathbf{x}' = \mathsf{MoE}(\mathbf{x})$.

The Mixture-of-Experts layer is an ensemble learning method that consists of two key components, Experts and Gating. Intuitively, it divides the input feature space into sub-spaces, and trains an Expert model for the feature alignment on each subspace. Then, given a new input vector $\mathbf{x}$, it utilizes the Gating model to decide which experts to use for $\mathbf{x}$. Formally, we use Expert_i and g_i to represent the i-th Expert and its gating weight. Then, we have

$$\mathbf{x}' = \mathsf{MoE}(\mathbf{x}) = (g_1 \cdot \mathsf{Expert}_1(\mathbf{x})) + \ldots + (g_k \cdot \mathsf{Expert}_k(\mathbf{x})). \tag{2}$$

We will discuss the challenges and our proposal on designing the above Experts and Gating models in Section 4.

Matcher: The Output Layer. Given the representation $\mathbf{x}'$, the `Matcher` is a binary classifier, which takes a vector $\mathbf{x}'$ as input and outputs its probabilities $\hat{y}$ of matching. For `Matcher`, MLP is the most common choice used in existing deep learning-based classifier (*e.g.,* DeepER [19] and Ditto [30]), which is denoted as $\hat{y} = \mathsf{M}(\mathbf{x}')$.

Multi-task Training. We adopt multi-task supervised learning to train **`Unicorn`**, using a lot of labeled match/non-match examples coming from all tasks in $\mathcal{T}$. Specifically, we union all the labeled element pairs in $\mathcal{T}$ to generate a training set $\mathcal{D} = \bigcup_i \mathcal{D}_i$, and train the above three modules of **`Unicorn`** in an end-to-end manner.

Data Matching Prediction. After multi-task training over all the tasks in $\mathcal{T}$, **`Unicorn`** can support the following prediction scenarios.

(1) Unified Prediction on Existing Tasks. In this scenario, we use **`Unicorn`** to predict the test set $\mathcal{D}_i^{\mathtt{tst}}$ of **any** task $T_i \in \mathcal{T}$. Note that $\mathcal{D}_i^{\mathtt{tst}}$ is *unlabeled* and disjoint with the training set $\mathcal{D}_i$ of T_i, *e.g.,* any unseen pairs for the existing tasks in Figure 3 (a).

(2) Zero-Shot Prediction on New Tasks. We can also use **`Unicorn`** to predict a **new** task T, which is not included in $\mathcal{T}$, with a *zero-shot* setting such that the pairs in the new task have **zero** labels. For example, given the learned models in Figure 3, we may use them to directly predict new unseen datasets of various task types (*e.g.,* string matching and column type annotation).

Remarks. We adopt the *blocking* technology [2, 14] to obtain appropriate labeled match/non-match examples for each task. For example, we can use simple string distance calculation rules, such as word overlapping, edit distance and euclidean distance, to filter out pairs that are unlikely matched. We also use a proportion of labeled match/non-match pairs as validation set, which is standard and thus not presented in this section for simplicity.

3 THE ENCODER MODULE

There are two main challenges in designing the `Encoder`. First, it is non-trivial to devise a *generic* serialization function $\mathsf{S}(\cdot)$ for various types of matching tasks over heterogeneous data elements. Second, existing works choose different pre-trained language models (PLMs) on individual matching tasks, *e.g.,* RoBERTa for entity matching [30] and BERT for entity alignment [46]. Thus, it remains an unresolved question on whether a *unified* PLM could achieve good performance on many different matching tasks.

Inspired by the recent successes of unified frameworks that treat many NLP tasks as a "text-to-text" problem [41, 54], we propose to serialize all data matching tasks into the *text* format and then utilize a *unified* PLM for encoding data element pairs in the tasks. To this purpose, in this section, we seek to answer two main questions: (1) which format should be used to unify different matching tasks; and (2) which *unified* PLM model should be utilized, so as to avoid the limitation that different PLMs have been used for different tasks.

3.1 Pair-to-Text Serialization

PLMs are natural choices for encoders, which typically take a sequence of tokens as input. Hence, we propose to serialize a pair of data elements (a, b) into a sequence of tokens (like Ditto [30]) as:

$$x = \mathsf{S}(a, b) = \texttt{[CLS]}\ \mathsf{S}(a)\texttt{[SEP]}\ \mathsf{S}(b)\ \texttt{[SEP]} \tag{3}$$

where `[CLS]` is a special token to indicate the start of the sequence, the first `[SEP]` is a special token to separate the sequence of element a (*i.e.,* $\mathsf{S}(a)$) and the sequence of element b (*i.e.,* $\mathsf{S}(b)$), and the last `[SEP]` token is used to indicate the end of the sequence.

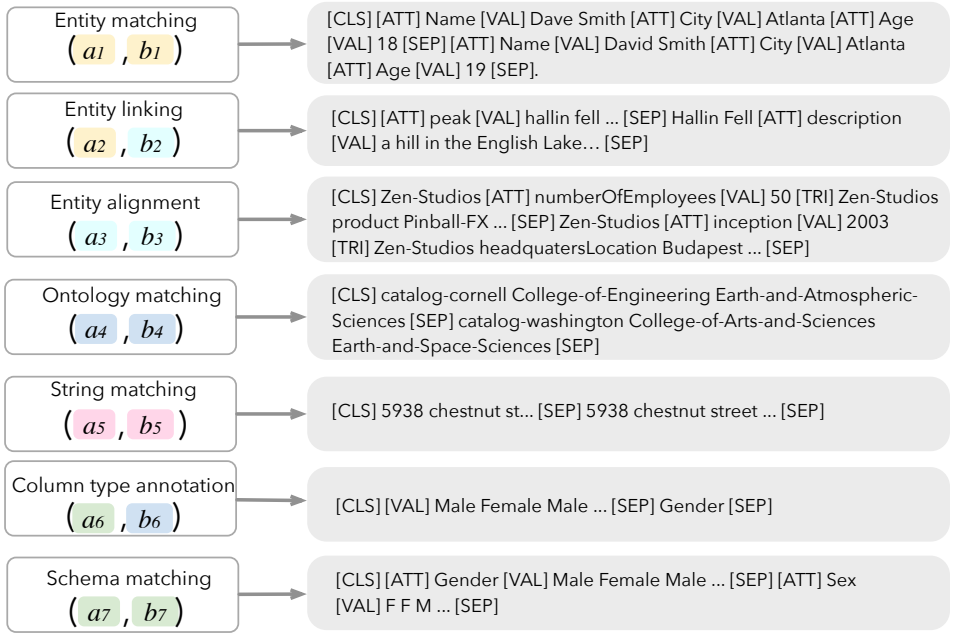


Fig. 4. A generic pair-to-text serialization that serializes pairs from data matching tasks into text sequences.

Next, we will describe how to serialize each type of the data elements.

String serialization. Given a string `str` with words $\langle \texttt{word}_i \rangle_{1 \le i \le k}$, we use the WordPiece tokenization algorithm, like BERT [11], to serialize `str` into a sequence of sub-words (tokens) as

$$\mathsf{S}(\texttt{str}) = \texttt{token}_1\ \texttt{token}_2\ \ldots\ \texttt{token}_{k'}.$$

For example, "*Hallin Fell is a hill in the English Lake District surrounded on three sides by Ullswater*" is serialized into the token sequence "Hall ##in Fell is a hill in the ##Eng ##lish Lake Dis ##trict su ##rround ##ed...". Note that we use this method to serialize all strings except special keywords in the following data elements.

Tuple serialization. Given a tuple `tup` with attribute-value pairs $\{(\texttt{attr}_i, \texttt{val}_i)\}_{1 \le i \le k}$, we serialize it into a sequence as

$$\mathsf{S}(\texttt{tup}) = \texttt{[ATT]}\ \texttt{attr}_1\ \texttt{[VAL]}\ \texttt{val}_1 \ldots \texttt{[ATT]}\ \texttt{attr}_k\ \texttt{[VAL]}\ \texttt{val}_k,$$

where `[ATT]` and `[VAL]` are two special tokens for specifying attributes and values respectively. Take tuple a_1 of entity matching in Figure 2 as an example: we serialize it into "`[ATT]` Name `[VAL]` Dave Smith `[ATT]` City `[VAL]` Atlanta `[ATT]` Age `[VAL]` 18".

Column serialization. Given a column `col` with an attribute name and values $(\texttt{attr}, \{\texttt{val}_i\}_{1 \le i \le k})$, we concatenate the attribute name and values of a whole column and serialize it as

$$\mathsf{S}(\texttt{col}) = \texttt{[ATT]}\ \texttt{attr}\ \texttt{[VAL]}\ \texttt{val}_1\ \texttt{val}_2 \ldots\ \texttt{val}_k.$$

Note that, in the case of too many values in the column, we randomly select a proportion of the values. Take the column a_7 of schema matching in Figure 2 as an example: we serialize a_7 into sequence "`[ATT]` Gender `[VAL]` Male Female Male …".

Ontology serialization. Given a tree-based ontology ont and a specific node_k in the ontology, we represent it as a sequence by concatenating all nodes in the path from root to node_k as

$$\text{S}(\text{node}_k) = \text{node}_1\ \text{node}_2 \ldots\ \text{node}_k.$$

Consider a_4 in Figure 2, the serialized sequence is "catalog-cornell College-of-Engineering Earth-and-Atmospheric-Sciences".

KG-entity serialization. Given a KG-entity, which is also denoted as a subject entity sub in the knowledge graph, it has not only some attribute values $\{(\text{attr}_i, \text{val}_i)\}_{1\le i\le k}$, but also some relational triples with other entities $\{\langle \text{sub}, \text{rel}_i, \text{obj}_i\rangle\}_{1\le i\le m}$. Based on the structure, we serialize the KG-entity sub into sequence

$$\begin{aligned}\text{S}(\text{sub}) = \text{sub}\ [\text{ATT}]\ \text{attr}_1\ [\text{VAL}]\ \text{val}_1 \ldots [\text{ATT}]\ \text{attr}_k\ [\text{VAL}]\ \text{val}_k \\ [\text{TRI}]\ \text{sub}\ \text{rel}_1\ \text{obj}_1 \ldots [\text{TRI}]\ \text{sub}\ \text{rel}_m\ \text{obj}_m,\end{aligned}$$

where [TRI] is a special token for specifying relational triples. As shown in Figure 2, we serialize a_3 as "Zen-Studios [ATT] numberOfEmployees [VAL] 50 [TRI] Zen-Studios product Pinball-FX [TRI] Zen-Studios location Hungary".

Figure 4 depicts the examples of serialized sequences for all data element pairs from the seven matching tasks as shown in Figure 2.

Zero-shot Instruction. For using **Unicorn** in *zero-shot* prediction on new tasks (see Section 2.2), we further utilize *instruction* [51, 52] to improve the performance. As a way to boost the inference ability of large-scale PLMs for downstream tasks in zero-shot/few-shot settings, instruction has been proven to be effective in many unified models, such as OFA [51], which specifies instruction templates described in natural language for a variety of multimodal tasks. SUPER-NATURALINSTRUCTIONS [52] integrates 1,616 natural language processing tasks into a unified framework using instruction and performs well on new tasks.

Inspired by these unified model, we develop a simple *task-agnostic* instruction for data mathing tasks in this paper. The basic idea is to define an appropriate natural language template to make downstream tasks (*e.g.,* matching tasks in our paper) conform to the natural language form of pre-training tasks. Specifically, we design the following simple instruction template, which is shown to be effective in our experiments (see Section 5 **Exp-5**).

$$x = \text{S}(a, b) = [\text{CLS}]\ \text{does}\ \text{S}(a)\ [\text{SEP}]\ \text{match with}\ \text{S}(b)\ [\text{SEP}] \tag{4}$$

Consider schema matching of (a_7, b_7) in Figure 4. By employing the above instruction template, the pair is serialized into a sequence "[CLS] does [ATT] Gender [VAL] Male Female Male ... [SEP] match with [ATT] Sex [VAL] F F M ... [SEP]".

Remarks. We only design a simple instruction template for matching tasks and preliminarily verify its performance on matching tasks in our experiments. We will discuss more instruction methods in the future work, such as specifying different instructions for different types of data matching tasks. In addition, prompt [29, 31, 54] is also an effective method to stimulate the ability of PLMs for downstream tasks, which will be systematically investigated in our future work.

3.2 Representation Learning of Serialized Pairs with Pre-trained Language Models

Given a sequence x serialized from pair (a, b), we employ a transformer-based pre-trained language models (PLM) as encoder to convert the sequence into a high-dimensional vector-based representation $\mathbf{x}$. PLMs have proven to be the most effective methods of learning representations.

Representative PLMs include BERT [11], RoBERTa [32], DeBERTa [24], etc. However, different from NLP tasks, the key challenge in **Unicorn** is to support *structure-aware encoding*, as tokens in

a serialized sequence may have specific structure information and depend on other tokens. For example, consider entity alignment over (a_3, b_3) in Figure 4, where the serialized sequence is

$S(a_3, b_3)$="[CLS] Zen-Studios [ATT] numberOfEmployees [VAL] 50 [TRI] Zen-Studios product Pinball-FX…".

In this example, "50" is a specific value, which highly depends on its attribute "numberofEmployees". Moreover, the token "Zen-Studios" after "[CLS]" plays more important roles in matching compared with the same token after "[TRI]", as the former is the name of the KG-entity a_3 to be matched and the latter just describes another entity connected to a_3. Thus, the former "Zen-Studios" should be paid more attention compared with the latter. Obviously, the conventional self-attention mechanism in Transformer has limitations to support the structure-aware encoding.

To address the problem, we employ DeBERTa [24] as the PLM for structure-aware encoding. The main reason is that DeBERTa has a new positional encoding scheme that captures *relative positions* of tokens in the sequence, which is helpful to understand the structure among the tokens. Consider our previous example again. The same token "Zen-Studios" will be encoded into different representations, as the first one is relatively near to "[CLS]", and thus should be attended by the entire sequence, while the second one only needs to be attended in the smaller scope of "[TRI]".

Specifically, DeBERTa [24] introduces *disentangled attention* and *enhanced mask decoder* to explicitly consider both relative and absolute positions of the tokens in an input sequence. The idea of *disentangled attention* is that, when calculating attention score (*i.e.,* relation score) between two tokens, DeBERTa considers not only the conventional content attention, but also an attention score based on relative positions. Formally, the attention score of two tokens at positions i and j in the sequence is calculated by the function (from the original DeBERTa paper [24]):

$$A_{i,j} = \{H_i, P_{i|j}\} \times \{H_j, P_{j|i}\}^T = H_i H_j^T + H_i P_{j|i}^T + P_{i|j} H_j^T + P_{i|j} P_{j|i}^T, \tag{5}$$

where H_i represents the content of token at the i-th position, and $P_{i|j}$ represents its relative position to the token at the j-th position. The above function shows that the attention between two tokens is computed by disentangled matrices, which consider both contents and positions as a sum of four attention scores: *content-to-content*, *content-to-position*, *position-to-content*, and *position-to-position*. Moreover, the *enhanced mask decoder* incorporates absolute position to context embedding right after all the transformer layers but before the `Softmax` layer during *Masked Language Modeling* (MLM) pre-training, which enables the model to understand absolute positions. Also, DeBERTa uses *virtual adversarial training* to improve the generalization of the model. More details can be found in the original paper of DeBERTa [24].

We also explore the benefits and limitations of a variety of PLMs, which are listed as follows.

- BERT [11] and RoBERTa [32] consider absolute positions of the input sequence, but do not explicitly model relative positions. Moreover, the MLM pre-training task used in BERT is also helpful for understanding absolute positions.
- XLNet [55] aims to capture relative positions by devising a pre-training task *Permuted Language Modeling*, which may damage the understanding of absolute positions.
- MPNet [45] considers both relative and absolute positions by *position compensation*.
- DistilRoBERTa [43], and DistilBERT [43] are some representative distillation models, which are smaller and faster to train.

Please refer to Section 5 **Exp-1** for a comprehensive comparison of different PLMs for **Unicorn**.

4 THE MIXTURE-OF-EXPERT MODULE

The objective of the `Mixture-of-Experts` (MoE) layer [21, 28, 44] in **Unicorn** is to map different distributions of multiple tasks to a same shared distribution. Thus, equipping **Unicorn** with MoE

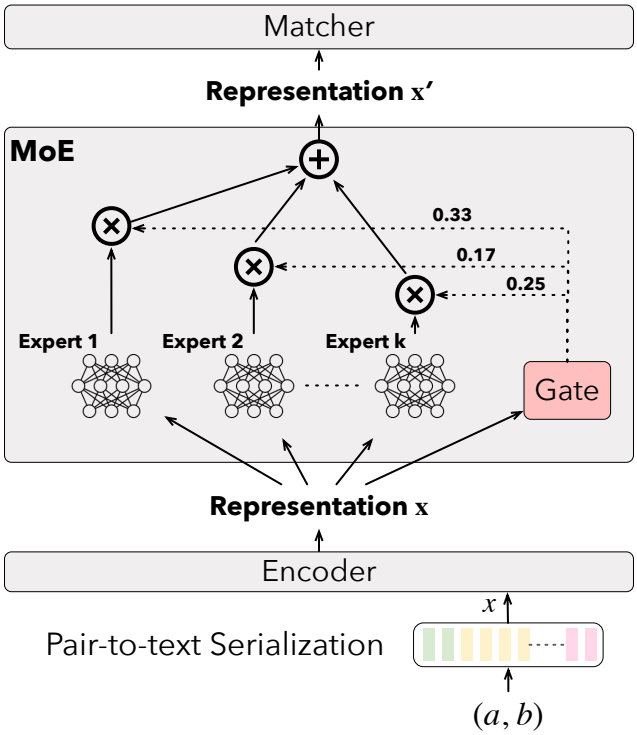


Fig. 5. An overview of the Mixture-of-Experts module. Experts are a set of neural networks to map $\mathbf{x}$ to different representations, and Gating is to combine the outputs of Experts according to learned weights that depend on the input $\mathbf{x}$.

will make it not only easily supporting multiple data matching tasks with different semantics and various input formats, but also being extensible to support new matching tasks.

Formally, the layer $\mathsf{MoE}(\mathbf{x}) : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$ is a function that converts an original feature vector $\mathbf{x} \in \mathbb{R}^{d_1}$ into a new feature vector $\mathbf{x}' \in \mathbb{R}^{d_2}$. In this paper, we adopt the most common Mixture-of-Experts architecture [25] that contains two key components, Experts and Gating. Experts are a set of neural networks, whose parameters are not shared, to individually map $\mathbf{x}$ to $\mathbf{x}'$ in the shared feature space. The aim of Gating is to combine the outputs of Experts according to different weights that depend on the input feature vector $\mathbf{x}$. Next, we first present the neural network design and the training algorithm for MoE in Section 4.1, and then introduce an optimization strategy for better routing among Experts in Section 4.2.

4.1 MoE Model Design and Training

Our neural network design for the Experts and Gating components in MoE is shown in Figure 5.

Neural Network Design. For each Expert_i, we use a fully-connected layer with LeakyReLU as activation function to convert the input feature vector $\mathbf{x}$ into an output $\mathbf{x}_i \in \mathbb{R}^{d_2}$:

$$\mathbf{x}_i = \mathsf{LeakyReLU}(\mathbf{x}W^{(i)}) \tag{6}$$

where $W^{(i)} \in \mathbb{R}^{d_1 \times d_2}$ are trainable parameters.

The Gating component also takes the feature vector $\mathbf{x}$ as input, and produces a gating vector $\mathbf{g} = (g_1, g_2, \ldots, g_k)$ where k is the number of Experts and g_k is the routing weight of Expert_i.

Specifically, we design two fully-connected layers with LeakyReLU and Softmax as activation functions, *i.e.,*

$$\mathbf{g} = \texttt{Softmax}\big(\texttt{LeakyReLU}(\mathbf{x}W_1^{\texttt{G}})W_2^{\texttt{G}}\big) \tag{7}$$

where $W_1^{\texttt{G}} \in \mathbb{R}^{d_1 \times h}$ and $W_2^{\texttt{G}} \in \mathbb{R}^{h \times k}$ are trainable parameters, h is dimension of the hidden layer, and k is the number of experts.

Based on Equations (6) and (7), we obtain k feature vectors, $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k$ through the k Experts, and then use weighted average to calculate a unified representation as $\mathbf{x}'$, *i.e.,*

$$\mathbf{x}' = \texttt{MoE}(\mathbf{x}) = g_1 \cdot \mathbf{x}_1 + g_2 \cdot \mathbf{x}_2 + \cdots + g_k \cdot \mathbf{x}_k. \tag{8}$$

Figure 5 illustrates an example of MoE. Given an encoded representation $\mathbf{x}$ from a pair (a, b), MoE divides the input feature space into k sub-spaces, each of which is handled with a trainable Expert model. It also devises a Gating model to compute k weights, *e.g.,* $(0.33, 0.17, \ldots, 0.25)$ from each individual input $\mathbf{x}$. Finally, MoE computes the weighted average of the outputs of the Experts.

End-to-End Model Training. We train the components Encoder, MoE and Matcher of **Unicorn** in an end-to-end manner, as presented previously. Specifically, the output $\mathbf{x}'$ of MoE is fed into Matcher to produce the predicted result $\hat{y} = \texttt{M}(\mathbf{x}')$. Given all labeled pairs $\mathcal{D} = \bigcup_i \mathcal{D}_i$ from different matching tasks (see Section 2.2), we can compute the cross entropy loss as

$$\mathcal{L} = \mathbb{E}_{(\mathbf{x}', y) \in \mathcal{D}} \mathcal{L}_{\texttt{CE}}(\hat{y}, y), \tag{9}$$

where $(\mathbf{x}', y)$ is a labeled matching/non-matching pair outputted by the previous Encoder and Mixture-of-Experts modules and $\mathcal{L}_{\texttt{CE}}$ is the cross entropy function.

Next, by iteratively applying minibatch stochastic gradient descent, parameters of the modules Encoder, MoE and Matcher are optimized, and thus **Unicorn** could be improved towards producing accurate matching results for different tasks.

4.2 MoE Optimization for Expert Routing

One obstacle of training MoE is that various input vectors $\{\mathbf{x}\}$ may use the same few Experts; that is, these Experts have larger gating weights than others for almost all the inputs. This phenomenon will affect the performance of MoE, as it fails to take advantage of different Experts for various data matching tasks.

To address this problem, we introduce an optimization strategy for Expert Routing. The basic idea is to improve the original loss function in Equation (9) by further considering two more objectives:

(1) Inspired by Sparsely-Gated MoE [44], for the overall training set, we want all the experts to be used in a *balanced* way.
(2) For any specific training pair, we would like to assign a few specific experts to it, instead of evenly assigning it to all the Experts, which is shown to achieve better performance in our experiments.

Technically, to achieve the first objective, we compute the "utilization" of all the experts on the overall training set $\mathcal{D}$. Formally, we compute an Expert utilization vector $\mathbf{u}$ for the k Experts as:

$$\mathbf{u} = (\sum_{\mathbf{x} \in \mathcal{D}} g_1, \sum_{\mathbf{x} \in \mathcal{D}} g_2, \ldots, \sum_{\mathbf{x} \in \mathcal{D}} g_k), \tag{10}$$

where $\sum_{\mathbf{x} \in \mathcal{D}} g_i$ is the sum of gating weights for $\texttt{Expert}_i$ on all the training examples in $\mathcal{D}$. As we want to achieve more balanced utilization of the Experts, we compute a *load balancing loss*, denoted as $\mathcal{L}_{\texttt{Bal}}$ by computing coefficient of variation of utilization $\mathbf{u}$ as:

$$\mathcal{L}_{\texttt{Bal}} = [\frac{\sigma(\mathbf{u})}{\mu(\mathbf{u})}]^2 \tag{11}$$

where $\sigma(\mathbf{u})$ and $\mu(\mathbf{u})$ are respectively standard deviation and mean of the utilization vector $\mathbf{u}$. Intuitively, the smaller the loss $\mathcal{L}_{\texttt{Bal}}$ is, the more balanced the utilization of the Experts is.

To achieve the second objective, we compute the *entropy* of the gating vector $\mathbf{g}$ for each training examples. The idea is that, we want that different Experts are trained to handle different feature sub-spaces, to make full use of the ensemble ability of MoE. Formally, we define an entropy loss function $\mathcal{L}_{\texttt{Ent}}$ over training examples:

$$\mathcal{L}_{\texttt{Ent}} = \mathbb{E}_{(\mathbf{x}',y)}\texttt{Entropy}(\mathbf{g}) = \mathbb{E}_{(\mathbf{x}',y)} - \sum_{i=1}^{k} g_i \cdot \log(g_i) \tag{12}$$

By considering the above loss functions, we derive a new loss function for training **Unicorn** as:

$$\mathcal{L}_{\texttt{new}} = \mathcal{L} + \mathcal{L}_{\texttt{Bal}} + \mathcal{L}_{\texttt{Ent}} \tag{13}$$

Note that, we apply the same minibatch stochastic gradient descent for training **Unicorn** given the new loss function. Specifically, in each training iteration, we compute the loss based on Equation (13) for training examples in a minibatch, and then update all the parameters in **Unicorn** by using back-propagation.

5 EVALUATION

Next, we report the experimental evaluation for **Unicorn**. The key questions we answer with our evaluation are presented as follows.

Exp-1: Which pre-trained language model should be employed as `Encoder` in **Unicorn**?
Exp-2: Which strategy performs well in the `Mixture-of-Experts` layer?
Exp-3: How does a unified model **Unicorn** (*i.e.,* trained using labeled datasets from multiple tasks) compare with specific state-of-the-art (SOTA) models (*i.e.,* each model is separately trained for ad-hoc tasks and datasets)?
Exp-4: How does **Unicorn** perform on a new task with zero labeled matching/non-matching pairs?
Exp-5: Whether our proposed MoE and instruction techniques are helpful in the zero-shot setting?
Exp-6: How does **Unicorn** perform for unseen tasks from totally new task types?

5.1 Experimental Setup

Datasets. Recall that **Unicorn** so far supports seven types of data matching tasks (see Figure 2). For each task type, we employ the commonly used datasets for evaluation, as shown in Table 1, each of which is denoted as a *task* (see definition of task in Section 2.2). Each task contains two sets, A and B, of data elements with different categories. Then, for each task, we derive the set $\mathcal{D}$ of labeled matching/non-matching pairs, and divide $\mathcal{D}$ into training $\mathcal{D}^{\texttt{trn}}$, validation $\mathcal{D}^{\texttt{val}}$ and test $\mathcal{D}^{\texttt{tst}}$. For fair comparisons of SOTA methods for each task, we prepare $\mathcal{D}^{\texttt{trn}}$, $\mathcal{D}^{\texttt{val}}$ and $\mathcal{D}^{\texttt{tst}}$ as follows.

(1) For each *entity matching* task, we directly use the training, validation and test sets prepared by the SOTA method Ditto [30], which have a ratio of 3:1:1, for a fair comparison with Ditto.

(2) For each *entity alignment* task, we follow the SOTA method BERT-INT [46] for data preparation. We first split the matching pairs for training, validation and test in a ratio of 2:1:7. Then, for each matching pair (a, b) for training and validation, we prepare one non-matching pair by replacing b with a randomly sampled data element $b' \in B$. For test set, we first find all matching pairs $\{(a, b)\}$, and then use all other combinations of $\{a\}$ and $\{b\}$ as non-matching pairs.

(3) For each task of *string matching*, *schema matching*, *ontology matching*, *column type annotation* and *entity linking*, the SOTA methods have not provided the prepared training, validation and test sets. Thus, we adopt blocking techniques to obtain appropriate labeled matching/non-matching examples. Specifically, we combine all data elements from A and B, and then apply some heuristic

Table 1. **Dataset Statistics, where $|A|$ (or $|B|$) represents the number of data elements in set A (or B) in each task, # Matches (# Non-Matches) represents the number of matches (non-matches) in the labeled set $\mathcal{D}$ of the task. Metric is the measurement for evaluating the corresponding data matching task.**

Task Type	Task	$\|A\|$	$\|B\|$	# Matches	# Non-Matches	Metric
Entity Matching (EM) (Tuple, Tuple)	Walmart-Amazon (WA)	2,554	22,074	962	9,280	F1
	DBLP-Scholar (DS)	2,616	64,263	5,347	23,360	F1
	Fodors-Zagats (FZ)	533	331	110	836	F1
	iTunes-Amazon (IA)	6,907	55,923	132	407	F1
	Beer (Be)	4,345	3,000	68	382	F1
Column Type Annotation (CTA) (Column, Ontology)	Efthymiou (Ef)	620	31	620	18,600	Acc.
	T2D (T2D)	383	37	383	13,788	Acc.
	Limaye (Lim)	174	27	179	4,519	Acc.
Entity Linking (EL) (Tuple, KG-Entity)	T2D (T2D)	11,650	26,025	20,666	131,945	F1
	Limaye (Lim)	659	4,166	1,447	36,020	F1
String Matching (StM) (String, String)	Address (Ad)	24,650	29,531	9,850	1,062	F1
	Names (Na)	10,341	15,396	5,132	2,763	F1
	Researchers (Re)	8,342	43,549	4,556	4,767	F1
	Product (Pr)	2,554	22,074	1,154	79,310	F1
	Citation (Ci)	2,616	64,263	5,347	34,152	F1
Schema Matching (ScM) (Column, Column)	FabricatedDatasets (Fa)	11,172	11,352	7,692	109,762	Recall
	DeepMDatasets (DM)	41	41	41	268	Recall
Ontology Matching (OM) (Ontology, Ontology)	Cornell-Washington (CW)	176	166	53	285	Acc.
Entity Alignment (EA) (KG-Entity, KG-Entity)	SRPRS: DBP-YG (SYG)	15,000	15,000	15,000	38,891	Hits@K
	SRPRS: DBP-WD (SWD)	15,000	15,000	15,000	38,492	Hits@K

blocking rules (such as having no common words or smaller string similarities) to filter out the pairs which are very likely to be non-matching, resulting in a candidate set. Then, we divide the candidate set into training, validation and test sets in a ratio of 2:1:7.

Evaluation Metrics. For fair comparisons with SOTA methods, we use the evaluation metrics which are also used by these methods.

(1) For *entity matching*, *entity linking*, and *string matching* tasks, following the SOTA methods, we use *F1* score as the evaluation metric. F1 score is the harmonic mean of precision and recall for the matching pairs, where precision P is the proportion of predicted true matching pairs to all predicted matching pairs, recall R is the proportion of predicted true matching pairs to all true matching pairs, and the F1 score is computed as $2 \cdot P \cdot R/(P + R)$.

(2) For *column type annotation* and *ontology matching* tasks, the SOTA methods use *accuracy* as the evaluation metric. By following them, we also use accuracy (Acc. for short), which is the ratio of the correct predicted pairs to all candidate pairs.

(3) For entity alignment, we use the common evaluation metric Hits@K, which is defined as the proportion of elements in A whose true matched elements in B are in the top-K matching results returned by an approach. Obviously, the higher the Hits@K is, the better an approach is. Following BERT-INT [46], we report Hits@1.

(4) For schema matching, like the SOTA method Valentine [27], we use Recall as the metric, which is the proportion of predicted matching schema pairs to all matching schema pairs.

Implementation Details of `Unicorn`. For `Encoder`, we explore the performance of BERT [11], RoBERTa [32], DeBERTa [24], MPNet [45], XLNet [55], DistilRoBERTa [43], and DistilBERT [43]. We use the pre-trained base size checkpoints directly on the Hugging Face [1]. For (BERT, RoBERTa,

Table 2. **Results for Representative Pre-trained Language Models for the Encoder Module.**

Type	Task	Metric	BERT	RoBERTa	DistilBERT	DistilRoBERTa	XLNet	MPNet	DeBERTa
EM	Walmart-Amazon	F1	84.24	84.37	75.07	68.99	79.43	83.01	**86.89**
	DBLP-Scholar	F1	**95.7**	94.88	93.92	95.53	95.62	95.02	95.64
	Fodors-Zagats	F1	97.67	95.24	92.68	95.24	**100**	97.67	**100**
	iTunes-Amazon	F1	94.55	96.3	90	87.1	89.66	94.55	**96.43**
	Beer	F1	87.5	**90.32**	86.67	**90.32**	84.85	**90.32**	**90.32**
CTA	Efthymiou	Acc.	**98.43**	98.1	97.51	97.68	97.58	98.13	98.42
	T2D	Acc.	**99.29**	98.75	98.22	98.27	98.17	98.89	99.14
	Limaye	Acc.	96.63	96.35	96.26	96.2	96.2	96.63	**96.75**
EL	T2D	F1	**93.65**	89.3	85.08	78.86	78.85	92.17	91.96
	Limaye	F1	85.08	85.8	66.71	78.94	82.53	83.37	**86.78**
StM	Address	F1	98.49	98.59	96.75	97.13	98.15	98.56	**98.68**
	Names	F1	92.58	**94.37**	55.47	90.71	76.26	80.97	91.19
	Researchers	F1	**98.99**	97.71	97.88	95.65	98.72	98.44	97.66
	Product	F1	81.13	80.67	65.8	63.71	71.73	76.81	**82.9**
	Citation	F1	**96.28**	95.7	93.74	95.11	95.91	95.28	96.27
ScM	FabricatedDatasets	Recall	77.85	77.72	42.96	77.48	70.99	70.62	**89.6**
	DeepMDatasets	Recall	88.89	88.89	92.59	92.59	**100**	88.89	96.3
OM	Cornell-Washington	Acc.	90.64	86.38	91.06	68.94	77.45	75.74	**92.34**
EA	SRPRS: DBP-YG	Hits@1	99.34	99.49	99.24	99.22	98.69	99.47	**99.67**
	SRPRS: DBP-WD	Hits@1	97.3	97.13	97.34	96.62	96.78	**97.47**	97.22
AVG			92.71	92.3	85.75	88.21	89.38	90.6	**94.21**

DeBERTa, MPNet, XLNet)-base models, they use 12 transformer layers and output a 768 dimensional hidden embedding. For DistilRoBERTa and DistilBERT, there are 6 transformer layers. We set the maximum sequence length as 128. For the MoE layer, we choose expert number from 2 to 15, and set hidden dimensions of gate and output size of experts from $\{384, 768, 1024\}$ according to the performance of validation set. A single fully connected layer is used for `Matcher` to output matching probabilities. We choose learning rate from {3e-5, 3e-6}, set batchsize as 32, and use maximum epoch number as 10.

Model Size. We compute the number of parameters of models to measure the model size. We use a python program to directly get the number of parameters. For `Encoder`, DeBERTa-base (used by **`Unicorn`**), RoBERTa-base (used by Ditto), BERT-base-multilingual (used by BERT-INT) and TinyBERT (used by TURL) have 139 Million (139M for shot), 125M, 178M and 14.5M parameters respectively. The `Mixture-of-Experts` layer has 8 Million parameters. We ignore the single layer in `Matcher` as it is too small. Thus, we compute the total model size for all methods by summing the numbers of parameters in the `Encoder` and the MoE layer (if any).

All the experiments are implemented using PyTorch [38] version 3.6.5 and the Transformers library [53], and evaluated on a server with 4 CPU cores (Intel Xeon Gold 6138 CPU @ 2.00GHz), 4 NVIDIA RTX 24GB GPUs, and 1024GB memory.

5.2 Evaluation on Unified Prediction

This section reports the experimental results for evaluating unified prediction (see Section 2.2). Specifically, we train a shared model **`Unicorn`** by multi-tasking training over all training sets from the data matching tasks, use the checkpoint of the model with the best average validation performance, and report the model performance on the test sets of different tasks.

Exp-1: Which pre-trained language model (PLM) should be employed in `Unicorn`? We first evaluate the performance of different PLMs. Under the same **`Unicorn`** framework, we use seven PLMs in base size as `Encoder`. Except for `Encoder`, we use the same settings and hyper-parameters to train the unified model. The results are reported in Table 2, where **bold** values are the best for

Table 3. **The overall Result for Unified Prediction. Unicorn w/o MoE is a variant of Unicorn that has no MoE layer. Unicorn is our proposed framework with Encoder, MoE and Matcher. Unicorn ++ is improved with MoE optimization for Expert Routing.**

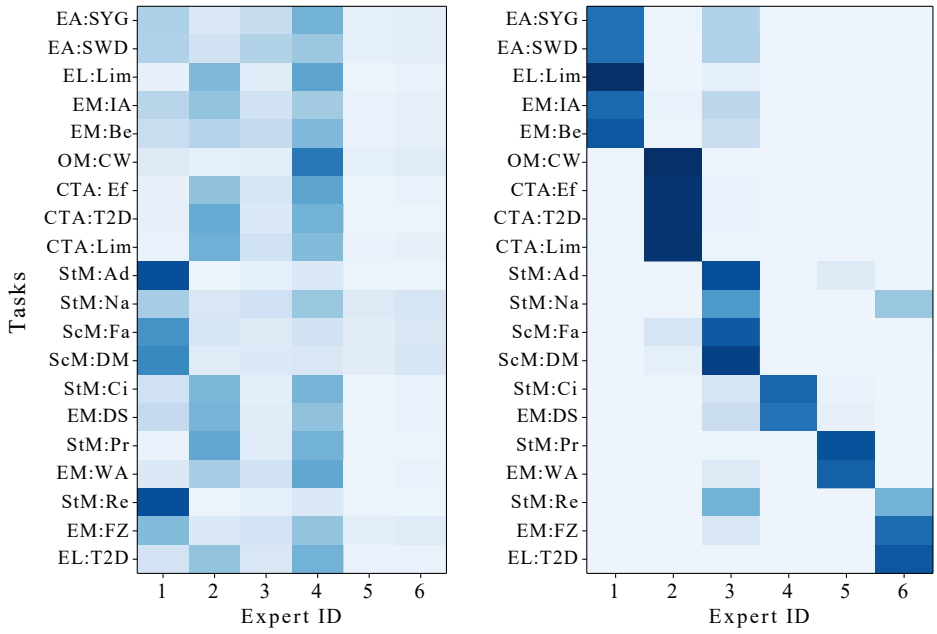
Type	Task	Metric	Unicorn w/o MoE	Unicorn	Unicorn ++	Previous SOTA (Paper)
EM	Walmart-Amazon	F1	85.12	86.89	**86.93**	86.76 (Ditto [30])
	DBLP-Scholar	F1	95.38	95.64	**96.22**	95.6 (Ditto [30])
	Fodors-Zagats	F1	97.78	**100**	97.67	**100** (Ditto [30])
	iTunes-Amazon	F1	94.74	96.43	**98.18**	97.06 (Ditto [30])
	Beer	F1	90.32	90.32	87.5	**94.37** (Ditto [30])
CTA	Efthymiou	Acc.	98.08	98.42	**98.44**	90.4 (TURL [10])
	T2D	Acc.	98.81	99.14	**99.21**	96.6 (HNN+P2Vec [5])
	Limaye	Acc.	96.11	96.75	**97.32**	96.8 (HNN+P2Vec [5])
EL	T2D	F1	79.96	91.96	**92.25**	85 (Hybrid I [20])
	Limaye	F1	83.12	86.78	**87.9**	82 (Hybrid II [20])
StM	Address	F1	97.81	98.68	99.47	**99.91** (Falcon [39])
	Names	F1	86.12	91.19	**96.8**	95.72 (Falcon [39])
	Researchers	F1	96.59	97.66	**97.93**	97.81 (Falcon [39])
	Product	F1	84.61	82.9	**86.06**	67.18 (Falcon [39])
	Citation	F1	96.34	96.27	**96.64**	90.98 (Falcon [39])
ScM	FabricatedDatasets	Recall	81.19	**89.6**	89.35	81 (Valentine [27])
	DeepMDatasets	Recall	66.67	96.3	96.3	**100** (Valentine [27])
OM	Cornell-Washington	Acc.	90.64	**92.34**	90.21	80 (GLUE [15])
EA	SRPRS: DBP-YG	Hits@1	99.46	99.67	99.49	**100** (BERT-INT [46])
	SRPRS: DBP-WD	Hits@1	97.11	97.22	97.28	**99.6** (BERT-INT [46])
AVG			90.8	94.21	**94.56**	91.84
Model Size			139M	147M	147M	995.5M

each task. We can see that **Unicorn** works well for all PLMs as Encoder, which proves the stability and generality of the framework. Also, we find that DeBERTa achieves the best performance on 11 tasks and on average, and performs as the second best in the remaining tasks. The main reason is that DeBERTa has a new positional encoding scheme that captures *relative positions* of tokens in the sequence, which is helpful to understand the structure among the tokens, as we discussed in Section 3.2. Furthermore, we find that the performance of DistilRoBERTa and DistilBERT are slightly worse than other base size models, mainly because distillation models sacrifice performance for time. Note that, in what follows, we will use DeBERTa by default.

Finding 1: Overall, we find that DeBERTa is the most suitable encoder for Unicorn, which requires structure-aware encoding and high generalization.

Exp-2: Which strategy performs well in Mixture-of-Experts of Unicorn? Table 3 reports the overall results for all tasks. **Unicorn** w/o MoE is a variant of **Unicorn** that has no MoE layer, *i.e.,* only using an Encoder and a Matcher. **Unicorn** is our proposed framework, which consists of an Encoder, a MoE intermediate layer, and a Matcher. **Unicorn** ++ is our proposed **Unicorn** improved with MoE optimization for Expert Routing (see Section 4.2).

We find that both **Unicorn** and **Unicorn** ++ are better than **Unicorn** w/o MoE, which means that MoE is helpful to our proposed unified architecture. In particular, **Unicorn** ++ achieves the best performance for most tasks and on average, which shows that Expert Routing can improve the overall performance. To provide an in-depth analysis, we visualize the average utilization weights of the Experts in each task using a heatmap shown in Figure 6 (the darker the color, the greater the weight). For **Unicorn**, Figure 6 (a) shows that all tasks mainly use the first four



(a) **Unicorn** (b) **Unicorn** ++

Fig. 6. **Visualization of average utilization weights of the** Experts **in each task. (a) For Unicorn with typical** MoE**, all tasks mainly use the first four** Experts**, while the weights of the** Experts **are even. (b) For Unicorn ++ with optimized** MoE**, the distinction of the** Experts **is more obvious.**

Experts, while the weights of the Experts are even. For **Unicorn** ++, Figure 6 (b) shows that the distinction of the Experts is more obvious. We find that similar tasks can be aggregated to the same expert. For example, all entity alignment (EA) tasks are (`KG-entity`, `KG-entity`) pairs and mainly aggregated to Expert 1. All column type annotation (CTA) are (`Column`, `Ontology`) pairs and aggregated to Expert 2. All schema matching (ScM) are (`Column`, `Column`) pairs and aggregated to Expert 3. For Expert 4 and Expert 5, the string matching (Stm) task StM:Ci and the entity matching (EM) task EM:DS are about bibliographies, while StM:Pr and EM:WA are about products. The above results show that our optimization technique that considers two new additional loss functions in Equation (13) is quite effective.

Finding 2: The MoE ***intermediate layer is quite helpful for*** **Unicorn**, ***while our*** MoE ***optimization for*** Expert ***Routing can further improve the overall performance.***

Exp-3: How does a unified model Unicorn (*i.e.,* trained using labeled datasets from multiple tasks) compare with specific models (*i.e.,* each model is separately trained for only one task)? We compare performance with the previous SOTA methods: Previous SOTA in Table 3 shows the results of the best task-specific model, which are reported by the existing papers, for each corresponding task. Because eight separated PLMs are used for BERT-INT [46], Ditto [30] and TURL [10], while other solutions use lightweight strategies without PLMs. For simplicity, we compute the model size of previous SOTA by just summing model sizes of the PLMs.

Table 4. **Zero-shot Performance of Unicorn (# of Labels is the number of labels needed by SOTA methods). Unicorn w/o MoE has no MoE layer. Unicorn has Encoder, MoE and Matcher. Unicorn-ins is improved with instruction.**

Type	Task	Metric	Unicorn w/o MoE	Unicorn	Unicorn-ins	SOTA (# of labels)
EM	DBLP-Scholar	F1	90.91	95.39	**97.08**	95.6 (22,965)
CTA	Limaye	Acc.	96.2	96.59	96.5	**96.8** (80)
EL	Limaye	F1	74.16	78.92	**82.8**	82 (-)
StM	Product	F1	60.71	74.92	**78.76**	67.18 (1,020)
ScM	DeepMDatasets	Recall	74.07	92.59	96.3	**100** (-)
EA	SRPRS: DBP-WD	Hits@1	95.55	97.25	96.17	**99.6** (4,500)
AVG			81.93	89.28	**91.27**	90.2

The experimental results show that our unified models (*i.e.,* **Unicorn** and **Unicorn** ++) outperform the previous SOTA methods on 15 over 20 tasks. For example, our unified models achieve the best accuracy on all tasks of column type annotation, significantly outperforming the previous SOTA method. Overall, our unified model **Unicorn** ++ gains an average evaluation score 94.56, while the previous SOTA is 91.84. Moreover, thanks to **task unification**, the model size of a unified **Unicorn** is much smaller, *i.e.,* 147M vs. 995.5M, where 147M is $139M + 8M$ for DeBERTa-base plus MoE layer and 995.5M is $125M \times 5 + 14.5M + 178M \times 2$ for 5 Ditto models, one TURL model, and two BERT-INT models in Previous SOTA (see Section 5.1 for calculation details of each model), compared with multiple specific models. The performance superiority of **Unicorn** is attributed to its **multi-task learning** that enables the unified model to learn from multiple tasks and multiple datasets to make full use of knowledge sharing. Specifically, as discussed in Exp-2, the data domains (*e.g.,* bibliographies, products, and so on) of entity matching and string matching tasks are similar, which enables **Unicorn** to have a better understanding of these specific data. Entity alignment and entity linking enable **Unicorn** to have a better understanding of knowledge graph entities. Column type annotation and schema matching have similar data elements (columns), which can improve the understanding ability of column data for **Unicorn**.

Finding 3: Our unified model achieves better performance on most datasets and on average, compared with the SOTA specific models trained for ad-hoc tasks and datasets separately.

5.3 Evaluation on Zero-Shot Prediction

Exp-4: How does Unicorn perform on unseen tasks with a zero-shot setting? This section explores the zero-shot effectiveness of our trained unified model **Unicorn**, *i.e.,* directly using the trained **Unicorn** to predict unseen new tasks without any labels. Specifically, for each task type, we randomly choose one task as new **unlabeled** task for testing, and only use the remaining tasks for multi-task training of **Unicorn**. As shown in Table 4, the results of **Unicorn** with **zero** label are comparable with previous SOTA methods with many labels. For example, for the entity matching task on DBLP-Scholar, the best variant of **Unicorn**, *i.e.,* **Unicorn-ins** achieves 97.08 on F1 score with zero label, outperforming 95.6 of Ditto [30] with 22, 965 labels. We also find that the overall performance of **Unicorn-ins** is better than that of previous SOTA methods, *i.e.,* 91.27 vs. 90.2. These results show that **Unicorn** can also well serve new matching tasks with zero-shot learning, as it has good matching knowledge sharing ability for new tasks. This also encourages us to release checkpoints of **Unicorn**, which are trained from multiple data matching tasks or multiple datasets for researchers and practitioners to use.

Finding 4:* Unicorn *can be effectively applied to new tasks, which are not included in the multi-task training, in a* zero-shot *setting such that pairs in the new task are unlabeled.

Exp-5: Whether our proposed MoE **and instruction techniques are helpful in the zero-shot setting?** We evaluate the performance of zero-shot learning on new tasks for three variants of **Unicorn**, namely **Unicorn** w/o MoE, **Unicorn** and **Unicorn-ins**. Table 4 shows the experimental results. We can see that **Unicorn** w/o MoE has the worst performance among the there variants. This result indicates that the MoE Layer is more conducive to preserving the performance of model on a new task, that is, to better use knowledge of seen tasks to predict on new unseen task. The main reason is that, the MoE layer learns from data instances what is the best way of aligning distributions between a new task and existing tasks, such that the models trained on the existing tasks can be used (or adapted) to the new task.

We also find that **Unicorn-ins** achieves the best performance, *i.e.,* outperforming the original **Unicorn** by about **2%** on average, as reported in Table 4. This result shows that the simple *instruction template* in Equation (4) is also quite promising in building one model to unify data matching tasks. There are some recent studies on exploring more sophisticated instruction or prompting techniques for matching tasks [36, 51, 52]. We will explore this direction as future work.

Finding 5: There is great potential for studying* Mixture-of-Experts *and instruction on* Unicorn *to further improve the performance of zero-shot predictions.

Exp-6: How does Unicorn perform for unseen tasks from totally new task types?

This section discusses the performance of **Unicorn** on completely new unseen task types, and we conduct experiments with the better improved **Unicorn** with instruction **Unicorn**-ins. The results in Table 5 are predicted by **Unicorn**-ins that are trained without the same task type. Specifically, for testing DBLP-Scholar of entity matching, the training data for **Unicorn**-ins does not contain any task from entity matching, for testing Limaya of column type annotation, the training data for **Unicorn**-ins does not contain any task from column type annotation, and the rest are similar. We find that **Unicorn**-ins is 1.79% lower than SOTA of specific solutions on average, the overall results of **Unicorn**-ins are acceptable, and **Unicorn**-ins is better than SOTA on Product of string matching that needs 1,020 labels (74.26 vs. 67.18). The results show that our unified model is comparable to the SOTA solution even if the unified model has never seen the same type of matching task and has no any specific label.

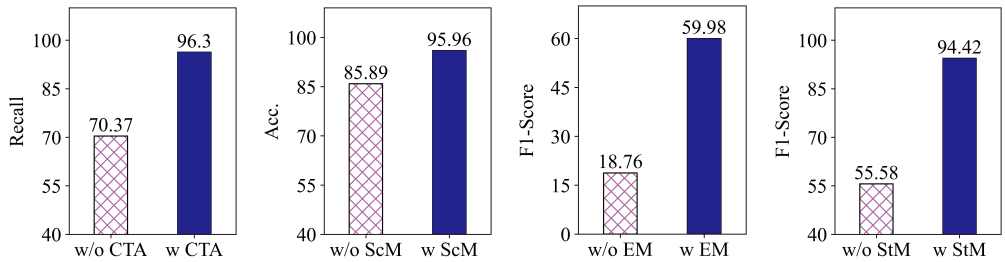
For a more in-depth analysis of the performance of **Unicorn** on new task types, we discuss the knowledge sharing capability of **Unicorn**. We control the tasks used for training **Unicorn**, and discuss how the addition of a certain type of task when training **Unicorn** affects the performance of test task. We consider the performance of **Unicorn** in two cases: similar task types and similar tasks.

(1) Similar task types. Matching task types involving the same data element type are similar task types. In this paper, both column type annotation and schema matching are about column element, while other task types are not. As shown in Figure 7(a), training **Unicorn** with column type annotation (CTA) tasks is helpful for the DeepMDatasets task of schema matching (96.3 vs. 70.37 on Recall). Similarly in Figure 7(b), using schema matching (ScM) tasks is also helpful for the Limaye task of column type annotation (95.96 vs. 85.89 on Acc.).

(2) Similar tasks. Matching tasks with the same data source are similar tasks. In this paper, we find that entity matching and string matching have similar tasks. For example, Walmart-Amazon and iTunes-Amazon in entity matching are about electronic products and music products, and DBLP-Scholar is about bibliographies, while string matching has corresponding products and bibliographies tasks (Product and Citation). Figure 7(c) shows that training **Unicorn** with entity

Table 5. **Zero-shot Performance of Unicorn for new unseen task types. (# of Labels is the number of labels needed by SOTA methods). Unicorn-ins is improved with instruction.**

Type	Task	Metric	Unicorn-ins	SOTA (# of labels)
EM	DBLP-Scholar	F1	94.5	95.6 (22,965)
CTA	Limaye	Acc.	96.23	96.8 (80)
EL	Limaye	F1	79.59	82 (-)
StM	Product	F1	**74.26**	67.18 (1,020)
ScM	DeepMDatasets	Recall	88.89	100 (-)
EA	SRPRS: DBP-WD	Hits@1	97	99.6 (4,500)
AVG			88.41	90.2



(a) ScM: DeepMDatasets (b) CTA: Limaye (c) StM: Product (d) EM: DBLP-Scholar

Fig. 7. Knowledge sharing capability of **Unicorn** for unseen new task types. (a) is the performance of DeepMDatasets in schema matching (ScM), w/o CTA corresponds to training **Unicorn** without using any column type annotation (CTA) tasks, while w CTA corresponds to training **Unicorn** with using column type annotation (CTA) tasks. The rest are similar.

matching (EM) tasks is helpful for improving the performance of Product in string matching (59.98 vs. 18.76 on F1). Figure 7(d) shows that string matching (StM) tasks are also helpful for improving the performance of DBLP-Scholar in entity matching (94.42 vs. 55.58 on F1).

The knowledge sharing capability of **Unicorn** is briefly discussed here, and we will discuss more in detail in future work to improve the robustness of **Unicorn** for various types of matching tasks.

Finding 6:* Unicorn *performs well on new unseen task types in a zero-shot setting. Training* Unicorn *with similar task types or similar tasks can improve the performance of new task types.

6 RELATED WORK

6.1 Data Matching Tasks

The "data matching" process is central to most, if not all, data integration problems. We consider the following common data matching tasks. *(1) Entity matching* discovers duplicate entities (or tuples), *i.e.,* those refer to the same real-world objects [7, 18]. Existing solutions define similarity functions based on attributes (*e.g.,* Magellan [14]) or use pre-trained language models to predict results as a binary classification task (*e.g.,* DeepER [19], Ditto [30], and DADER [48])). *(2) Entity linking* [10] is a task of determining if a reference entity in a knowledge base is the potential mention of a tuple

within table. Existing solutions typically retrieve potential entity mentions from knowledge bases and then sort them by calculating their similarities (*e.g.,* Hybrid II [20]). *(3) Entity alignment* refers to the task of identifying equivalent entities across different knowledge graphs (KGs) [56]. Existing solutions mainly learn entity embedding and realize the matching of embedding through graph neural networks (*e.g.,* GCN [26]) or pre-trained language models (*e.g.,* BERT-INT [46]). *(4) String matching* [37] decides whether two strings are equivalent, either synthetically or semantically. Existing solutions use string similarity functions [50] or machine learning methods such as decision tree to predict the results (*e.g.,* Smurf [39], Falcon [8]). *(5) Column type annotation* decides semantic type (*e.g.,* Gender) of a column (*e.g.,* {male, female, female, . . .}). Existing solutions use a single cell embedding or column embedding to represent a column through neural networks and then determine its type by similarity function or machine learning, where the type is a string or category (*e.g.,* HNN+P2Vec [5], TURL [10]). *(6) Schema matching* aims at finding the correspondence between schemas across different tables [27]. Existing solutions commonly define heuristic rules or calculate similarities between schemas (*e.g.,* COMA [12]). *(7) Ontology matching* [16] is the problem of finding semantic mappings between two given ontologies. Existing solutions calculate similarities such as *Jaccard* similarity cross different nodes of ontology and use machine learning to predict the results (*e.g.,* GLUE [15]).

Although these problems have been studied for several decades, because they lie at the heart of data integration with almost all applications, they remain to be important. Existing approaches try to solve each problem separately, *e.g.,* using different frameworks or different training methods. Moreover, these solutions are not only task-specific, but also oftentimes dataset-specific (*e.g.,* for each new task, saying entity matching between Walmart and Amazon products, we have to re-train the model only for serving this dataset). Different from them, **Unicorn** aims at a unified matching model that can be used for multiple matching tasks and datasets out-of-the-box, such as GPT-3 [3] for a general language model.

6.2 Transformer-based Language Models

The paper "Attention Is All You Need" [49] introduces a novel architecture called Transformer, which uses the attention mechanism for sequence to sequence learning. Transformer was originally proposed for natural language processing. Recently, Transformer has shown good performance for images [17], as well as for many modalities such as DeepMind Gato [42]. In addition, Transformer-based language models such as BERT [11] and RoBERTa [32] have also been used to deal with some database tasks (such as PASTA [23] and SCPromt [22] for Text-to-SQL translation, and Symphony [6] for querying data lakes) and data integration tasks (such as Ditto [30] for entity matching and DeepBlocker [47] for the blocking task in entity matching).

Inspired by these recent successes using Transformer-based models in a wide range of applications, in this paper, we investigate the possibility of training a unified model that is generally capable of supporting a large number of matching tasks, which is crucial to data integration and data management, but is not well explored.

6.3 Mixture-of-Experts Models

Mixture-of-Experts [21, 25, 28, 44] (abbreviated as MoE) is an ensemble learning technique that implements the idea of training multiple experts on sub-tasks. Generally speaking, MoE trains each expert to examine a different part of the space (*e.g.,* different tasks of input data in our problem). A gating network is responsible for combining various experts. Recently, MoE has been widely used in big tech companies such as Google, Microsoft, and Facebook.

MoE is a good fit to **Unicorn** for two reasons. First, **Unicorn** aims at supporting multiple data matching tasks with different semantics and various input formats, for which each expert can focus

on learning matching relevant to specific data inputs. Second, equipping **Unicorn** with MoE will make it easily extensible, *e.g.,* to support a new matching task between a tuple and an image (see our discussion in Section 7).

7 CONCLUSION AND FUTURE WORK

We have proposed **Unicorn**, the first unified model for supporting multiple data matching tasks. So far, **Unicorn** supports seven data matching tasks over five different types of data elements. This unified model can enable **knowledge sharing** by learning from multiple tasks and multiple datasets, and also support **zero-shot prediction** for new tasks with zero labeled pairs. We develop a general framework for **Unicorn** that employs an `Encoder`, a `Mixture-of-Experts` and a `Matcher`. We conduct experiments on 20 datasets of the seven matching tasks. Experimental results show that **Unicorn** is not only comparable or even better than SOTA specific models, but also well performs zero-shot learning on unseen new tasks.

Unicorn still has a lot of potential to expand. The optimized MoE and instruction we discussed lead to obvious improvements, and we will continue to explore more possibilities in the future, *e.g.,* using data augmentation techniques to synthesize new labeled data, to reduce human annotation cost and improve model performance. Another interesting future work is to extend **Unicorn** to support more modalities (*e.g.,* images), such as whether a picture matches a person (*e.g.,* Michael Jordan) in a knowledge graph. As data matching tasks are typically supervised, for the tasks **Unicorn** already supports and those that we plan to support as discussed above, another immediate future work is to collect more labeled data and enrich our current benchmark.

ACKNOWLEDGMENTS

This work was partly supported by the NSF of China (U1911203, 62122090, 62072461, 62072458, 62232009, and 61925205), National Key Research and Development Program of China (2020YFB2104101), Huawei, TAL education, and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

[1] 2020. *HuggingFace Datasets.* https://huggingface.co/datasets
[2] Fabio Azzalini, Songle Jin, Marco Renzi, and Letizia Tanca. 2021. Blocking techniques for entity linkage: A semantics-based approach. *Data Science and Engineering* 6 (2021), 20–38.
[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
[4] Chengliang Chai, Nan Tang, Ju Fan, and Yuyu Luo. 2023. Demystifying Artificial Intelligence for Data Preparation. *Proceedings of the 2023 ACM SIGMOD international conference on Management of data.*
[5] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019. Learning semantic annotations for tabular data. *arXiv preprint arXiv:1906.00781* (2019).
[6] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. In *2023 Conference on Innovative Data Systems Research (CIDR).*
[7] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–42.
[8] Sanjib Das, Paul Suganthan GC, AnHai Doan, Jeffrey F Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *Proceedings of the 2017 ACM International Conference on Management of Data.* 1431–1446.
[9] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings.*
[10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proceedings of the VLDB Endowment* 14, 3 (2020), 307–319.

[11] 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).
[12] Hong Hai Do and Erhard Rahm. 2002. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*. Morgan Kaufmann, 610–621.
[13] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration*. Elsevier.
[14] AnHai Doan, Pradap Konda, Paul Suganthan GC, Yash Govind, Derek Paulsen, Kaushik Chandrasekhar, Philip Martinkus, and Matthew Christie. 2020. Magellan: toward building ecosystems of entity matching solutions. *Commun. ACM* 63, 8 (2020), 83–91.
[15] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. 2003. Learning to match ontologies on the semantic web. *The VLDB journal* 12, 4 (2003), 303–319.
[16] AnHai Doan, Jayant Madhavan, Pedro M. Domingos, and Alon Y. Halevy. 2004. Ontology Matching: A Machine Learning Approach. In *Handbook on Ontologies*. Springer, 385–404.
[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
[18] Halbert L. Dunn. 1946. Record Linkage. *American Journal of Public Health* 36, 12 (1946), 1412–1416.
[19] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.
[20] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *International Semantic Web Conference*. Springer, 260–277.
[21] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. , 5232–5270 pages.
[22] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Samuel Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proceedings of the 2023 ACM SIGMOD international conference on Management of data*.
[23] Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-Operations Aware Fact Verification via Sentence-Table Cloze Pre-training. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*. Association for Computational Linguistics, 4971–4983.
[24] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654* (2020).
[25] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.
[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[27] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 468–479.
[28] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).
[29] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).
[30] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment* 14, 1 (2020), 50–60.
[31] 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).
[32] 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).
[33] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1930–1939.
[34] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In

Proceedings of the 2018 International Conference on Management of Data. 19–34.

[35] Hannes Mühleisen and Christian Bizer. 2012. Web Data Commons-Extracting Structured Data from Two Large Web Corpora. *LDOW* 937, 133–145.

[36] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *CoRR* abs/2205.09911 (2022). arXiv:2205.09911

[37] Gonzalo Navarro. 2001. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.* 33, 1 (mar 2001), 31–88. https://doi.org/10.1145/375360.375365

[38] 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. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[39] GC Paul Suganthan, Adel Ardalan, AnHai Doan, and Aditya Akella. 2019. Smurf: Self-service string matching using random forests. *Proceedings of the VLDB Endowment* 12, 3 (2019).

[40] Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. 2020. Multitask mixture of sequential experts for user activity streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3083–3091.

[41] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[42] Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A Generalist Agent. *CoRR* abs/2205.06175 (2022).

[43] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[44] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[45] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems* 33 (2020), 16857–16867.

[46] Xiaobin Tang, Jing Zhang, Bo Chen, Yang Yang, Hong Chen, and Cuiping Li. 2021. BERT-INT: a BERT-based interaction model for knowledge graph alignment. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 3174–3180.

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

[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 *Proceedings of the 2022 International Conference on Management of Data*. 443–457.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30.

[50] Jiannan Wang, Jianhua Feng, and Guoliang Li. 2010. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1219–1230.

[51] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*. PMLR, 23318–23340.

[52] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 5085–5109.

[53] 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).

[54] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966* (2022).

[55] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[56] Kaisheng Zeng, Chengjiang Li, Lei Hou, Juanzi Li, and Ling Feng. 2021. A comprehensive survey of entity alignment for knowledge graphs. *AI Open* 2 (2021), 1–13.
[57] Xiang Zhao, Weixin Zeng, Jiuyang Tang, Xinyi Li, Minnan Luo, and Qinghua Zheng. 2022. Toward Entity Alignment in the Open World: An Unsupervised Approach with Confidence Modeling. *Data Science and Engineering* 7, 1 (2022), 16–29.

Received July 2022; revised October 2022; accepted November 2022