Unicorn: A Unified Multi-Tasking Matching Model

Ju Fan¹, Jianhong Tu¹, Guoliang Li², Peng Wang¹, Xiaoyong Du¹
Xiaofeng Jia³, Song Gao³, Nan Tang⁴
¹Renmin University of China, Beijing, China ²Tsinghua University, Beijing, China
³Beijing Big Data Centre, Beijing, China ⁴HKUST (GZ) / HKUST, Guangzhou, China
fanj@ruc.edu.cn

ABSTRACT

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. 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. Building such a unified model is challenging due to heterogeneous formats of input data elements 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 adopts a mixture-of-experts model that enhances the learned representation into a better representation. We conduct extensive experiments using 20 datasets on 7 well-studied data matching tasks, 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.

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, etc. Data matching, as a core concept in data integration [10] and data preparation [7], includes a wide spectrum of tasks. In this paper, we consider common data matching tasks, including entity matching, entity linking, entity alignment, string matching, column type annotation, schema matching, and ontology matching.

Due to their importance, almost all the aforementioned matching tasks have been extensively studied in both database and machine learning communities, and still remain to be important research topics. Traditional wisdom to solve the matching tasks is to build task-specific or even dataset-specific (machine learning) models [27, 24, 8, 42], which are referred to as specific models. However, these specific models do not facilitate knowledge sharing across different tasks or datasets. For example, the knowledge learned by a column type annotation model \((a.g., \text{TURL} [8])\) cannot be shared with an entity matching model \((a.g., \text{Ditto} [24])\) due to different neural network designs, although very likely the two models can help each other. Moreover, training (or fine-tuning) each model for each task or dataset usually needs a huge amount of labeled data, which is time and effort consuming to obtain. For example, DeepMatcher [27] and Ditto [24] have to be fine-tuned on a new training dataset with 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 the previous specific models, Unicorn has the following key innovations.

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** [44, 6] 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** [31, 40] 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 machine learning community [1, 33, 34], building such a unified model for supporting multiple data matching tasks is challenging. Firstly, 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. Secondly, each data matching task may have its unique matching semantics, making it non-trivial to enable knowledge sharing among multiple tasks.
2. PROBLEM: DATA MATCHING TASKS

2.1 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 five types of data elements.

String. A string is a sequence of words, which could be a noun or a natural language sentence, denoted as \(\langle\text{word}_1\rangle_{1\leq i\leq k}\).

Tuple. A tuple is a row contained in a table, consisting of a set of attribute-value pairs \(\{\langle\text{attr}_i, \text{val}_i\rangle\}_{1\leq i\leq k}\), where attr\(_i\) and val\(_i\) are respectively the \(i\)-th attribute name and value of the tuple.

Column. A column is a set of values \(\{\text{val}_i\}_{1\leq i\leq k}\) of a particular attribute attr\(_i\) 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 \(\text{ont} = \{\text{node}, \text{parent}_{\text{node}}\}_{1\leq i\leq k}\), where parent\(_{\text{node}}\) 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 \(\text{KG} = (E, R, A, V, T, P_n)\), where \(\text{ent} \in E\), \(\text{rel} \in R\), \(\text{attr} \in A\), and \(\text{val} \in V\) represent an entity, a relation, an attribute, and an attribute value respectively. Moreover, \((\text{ent}_a, \text{rel}_a, \text{ent}_b) \in T\); \((\text{ent}_a)\) denotes a relational triple, and \((\text{attr}_a, \text{val}_a) \in P_n\) denotes an attribute-value pair.

2.2 Data Matching

Let \(A = \{a_1, \ldots, a_n\}\) and \(B = \{b_1, \ldots, b_n\}\) 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 depends on the specific data matching tasks. We consider the following seven common types of data matching tasks. For each task, we describe the existing solutions and our formal definitions.

(1) Entity matching refers to the task of determining whether two different tuples from two tables refer to the same real-world object [13, 5], which is formalized as determining whether a (Tuple, Tuple) pair matches or not.

(2) Entity linking refers to the task of determining whether a mention in a table refers to the same object with a mention in another table, i.e., a unification of data integration. A mention is represented by a tuple that contains multiple attributes. Formally, given a pair of (Tuple, KG-Entity), one needs to deduce whether they are matched or not.

(3) Entity alignment refers to a task of determining whether a pair of two KG-entities (KG-Entity, KG-Entity), typically from different knowledge graphs (e.g., DBPedia and YAGO), are the same real-world object.

(4) String matching refers to the task of determining whether two strings (String, String) from two data sources are semantically the same or not.

(5) Column type annotation determines the semantic types of a column in a table, which is formalized as determining whether a (Column, Ontology) pair matches or not.
Schema matching determines the correspondences between columns of two schemata from different tables. A typical schema matching solution requires to determining whether a (Column, Column) pair matches or not.

Ontology matching finds correspondences between semantically related entities from different ontologies, formalized as determining if an (Ontology, Ontology) pair matches.

3. A UNIFIED MATCHING FRAMEWORK

3.1 An Overview of Unicorn

Unicorn offers a unified approach to multi-tasking data matching, characterized by two main advantages:

1. Task unification standardizes task-specific solutions into a unified framework, achieving lower development complexity, smaller model sizes and easier adaptation of new tasks.

2. Multi-task learning [44, 6] enables the possible opportunities of knowledge sharing among different data matching tasks compared with training each task separately.

As depicted in Figure 2, Unicorn unifies multiple data matching tasks into a text-to-prob format, offering flexibility and extensibility in handling diverse tasks. It achieves this through input serialization of any data element pair \((a, b)\) into a text sequence and outputs a probability \(y\) indicating the likelihood of a match. Unicorn comprises three core modules: Encoder, Mixture-of-Experts, and Matcher, each playing a crucial role in the unified data matching process.

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 \(T = \{T_i\}\). In particular, each task \(T_i = (A_i, B_i, \tau_i, D_i)\) is composed of two sets of data elements to be matched, i.e., \(A_i\) and \(B_i\), and \(\tau_i\) is the type of task \(T_i\) (see Section 2 for the supported task types). \(D_i \subset A_i \times B_i \times \{0, 1\}\) 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).

Encoder: The Input Layer. Given an element pair \((a, b)\) from any data matching task (e.g., Figure 2 (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 \(\hat{x}\), i.e.,

\[
x = F(x) = F(S(a, b)),
\]

where \(S(\cdot)\) is a generic function for serializing any data element pair \((a, b)\) from the matching tasks in \(T\) into a text sequence, and \(F(\cdot)\) is a pre-trained language model (PLM) for deriving high-dimensional feature vectors from the serialized sequences. See Section 3.2 for details.

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 2 (b). Consequently, it is hard to train a good Matcher.

To address the problem, we introduce an intermediate layer Mixture-of-Experts (MoE) [26, 35, 32, 22] to align the representations of different tasks, such that a good Matcher is easier to learn, as shown in Figure 2 (c).

3.2 The Encoder Module

Our approach is inspired by recent developments in unified NLP frameworks. We propose serializing all data matching tasks into text format and then using a unified PLM for encoding. This section aims to determine the optimal format for unifying different matching tasks and identify the most effective unified PLM, overcoming the limitations of using different PLMs for various tasks.

3.2.1 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 [24]) as:

\[
x = S(a, b) = [CLS] S(a) [SEP] S(b) [SEP]
\]

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., \(S(a)\)) and the sequence of element \(b\) (i.e., \(S(b)\)), and the last [SEP] token is used to indicate the end of the sequence.

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

String serialization. Given a string \(\text{str}\) with words \(\langle\text{word}\rangle_{1 \leq i \leq k}\), we use the WordPiece tokenization algorithm, like BERT [9], to serialize \(\text{str}\) into a sequence of sub-words (tokens) as \(S(\text{str}) = \{\text{token}\}_1 \ldots \{\text{token}\}\).

Tuple serialization. Given a tuple \(\text{tup}\) with attribute-value pairs \(\{(\text{attr}_i, \text{val}_i)\}_{1 \leq i \leq k}\), we serialize it into a sequence as \(S(\text{tup}) = [\text{ATT}] \text{attr}_1 \{\text{VAL}} \text{val}_1 \ldots [\text{ATT}] \text{attr}_k \{\text{VAL}} \text{val}_k\), where [ATT] and [VAL] are two special tokens for specifying attributes and values respectively.

Column serialization. Given a column \(\text{col}\) with an attribute name and values \(\{(\text{attr}, \{\text{val}\})\}_{1 \leq i \leq k}\), we concatenate the attribute name and values of a whole column and serialize it
as $S(col) = [ATT] \text{attr} [VAL] val_1, val_2, ..., val_k$. Note that, in the case of too many values in the column, we randomly select a proportion of the values.

**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 $S(node_k) = node_0, node_1, ..., node_k$.

**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 $\{(\text{attr}_i, \text{val}_i)\}_{1 \leq i \leq k}$, but also some relational triples with other entities $\{(\text{rel}_i, \text{obj}_i)\}_{1 \leq i \leq m}$. Based on the structure, we serialize the KG-entity sub into sequence $S(sub) = [ATT] \text{attr}_1 [VAL] val_1, ..., [ATT] \text{attr}_k [VAL] val_k [TRI] \text{sub rel}_1 \text{obj}_1, ..., [TRI] \text{sub rel}_m \text{obj}_m$, where $[TRI]$ is a special token for specifying relational triples.

Figure 3 depicts the examples of serialized sequences for all data element pairs from the seven matching tasks.

**Zero-shot Instruction.** For using Unicorn in zero-shot prediction on new tasks, we further utilize instruction [39, 41] to improve the performance. Specifically, we develop a simple task-agnostic instruction for data matching tasks in this paper. The basic idea is to define an appropriate natural language template to make downstream matching tasks conforming to the natural language form of pre-training tasks.

### 3.2.2 Representation Learning of Serialized Pairs with Pre-trained Language Models

Representative PLMs include BERT [9], RoBERTa [25], and DeBERTa [19]. A key challenge in Unicorn is to support structure-aware encoding, where tokens in serialized sequences carry specific structural information. For example, in entity alignment, the serialized sequence $S(a_3, b_3) = [CLS] Zen-Studios [ATT] numberOfEmployees [VAL] 50 [TRI] Zen-Studios product Pinball-FX” demonstrates this dependency and importance of token positions.

Conventional self-attention in Transformers may not perform well for structure-aware encoding. Thus, we use DeBERTa [19] for its positional encoding scheme that captures relative token positions. DeBERTa’s disentangled attention mechanism, as shown in Equation (3), calculates attention scores considering both contents and relative positions of tokens:

$$A_{i,j} = \{H_i, P_{i,j}\} \times \{H_j, P_{j,i}\}^T = H_i H_j^T + H_i P_{i,j}^T + P_{i,j} H_j^T + P_{i,j} P_{j,i}^T, \quad (3)$$

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.

### 3.3 The Mixture-of-Expert Module

The Mixture-of-Experts (MoE) layer’s objective in Unicorn [35, 16, 22] is to map various task distributions to a shared distribution. By integrating MoE, Unicorn supports multiple data matching tasks with diverse semantics and formats, and can be adapted to new tasks.

Formally, $MoE(x) : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$ transforms an original feature vector $x \in \mathbb{R}^{d_1}$ to a new vector $x’ \in \mathbb{R}^{d_2}$. The common
The output weighted average to calculate a unified representation as vectors, of experts.

\[ \text{vector} \]

\[ \text{Neural Network Design.} \]

For expert routing is discussed in Section 3.3.2.

are detailed in Section 3.3.1, and an optimization strategy for expert routing is discussed in Section 3.3.2.

**3.3.1 MoE Model Design and Training**

Our neural network design for the Experts and Gating in MoE is detailed in Figure 4.

**Neural Network Design.** Each Expert uses a fully-connected layer with LeakyReLU activation to map the input feature vector \( \mathbf{x} \) to \( \mathbf{x}_i \), as shown in Equation (4):

\[ \mathbf{x}_i = \text{LeakyReLU}(\mathbf{x} W_i^{(1)}) \]  

where \( W_i^{(1)} \in \mathbb{R}^{d_1 \times d_2} \) are trainable parameters.

The Gating component uses two fully-connected layers with LeakyReLU and Softmax activations to produce a gating vector \( \mathbf{g} \), as depicted in Equation (5):

\[ \mathbf{g} = \text{Softmax}(\text{LeakyReLU}(\mathbf{x} W_1^{(2)})) W_2^{(2)} \]  

where \( W_1^{(2)} \in \mathbb{R}^{d_1 \times h} \) and \( W_2^{(2)} \in \mathbb{R}^{h \times k} \) are trainable parameters, \( h \) is dimension of the hidden layer, and \( k \) is the number of experts.

Based on Equations (4) and (5), we obtain \( k \) feature vectors, \( \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \), through the \( k \) Experts, and then use weighted average to calculate a unified representation as \( \mathbf{x}' \):

\[ \mathbf{x}' = \text{MoE} (\mathbf{x}) = g_1 \cdot \mathbf{x}_1 + g_2 \cdot \mathbf{x}_2 + \cdots + g_k \cdot \mathbf{x}_k. \]  

**End-to-End Model Training.** The Encoder, MoE, and Matcher components of Unicorn are trained end-to-end. The output \( \mathbf{x}' \) of MoE is input to Matcher for predictions \( \hat{y} = M(\mathbf{x}') \). Loss is computed using cross entropy function \( L_{CE} \) over labeled matching/non-matching pairs.

**3.3.2 MoE Optimization for Expert Routing**

Training MoE faces the challenge of input vectors \( \{\mathbf{x}\} \) favoring a few Experts over others, affecting performance. To tackle this, we introduce an optimization strategy for Expert routing, addressing two objectives: (1) ensuring balanced use of all Experts across the training set, inspired by Sparsely-Gated MoE [35], and (2) assigning specific experts to specific training pairs for better performance.

To achieve balanced utilization, we calculate an Expert utilization vector \( \mathbf{u} \) (Equation (7)) and a load balancing loss \( L_{bal} \), which is the squared coefficient of variation of \( \mathbf{u} \):

\[ \mathbf{u} = \left( \sum_{x \in \mathcal{D}} g_1, \sum_{x \in \mathcal{D}} g_2, \ldots, \sum_{x \in \mathcal{D}} g_k \right), \]  

where \( \sum_{x \in \mathcal{D}} g_i \) is the sum of gating weights for Expert \( i \) on all the training examples in \( \mathcal{D} \).

\[ L_{bal} = \frac{\sigma(\mathbf{u})^2}{\mu(\mathbf{u})} \]

where \( \sigma(\mathbf{u}) \) and \( \mu(\mathbf{u}) \) are respectively standard deviation and mean of the utilization vector \( \mathbf{u} \).

For the second objective, we calculate the entropy of the gating vector \( \mathbf{g} \) for each training example, defining an entropy loss function \( L_{ent} \):

\[ L_{ent} = \mathbb{E}(x,y) \sum_{i=1}^{k} g_i \cdot \log(g_i) \]

The new loss function for training Unicorn is:

\[ L_{new} = L + L_{bal} + L_{ent} \]

**4. EXPERIMENTS**

**4.1 Experimental Setup**

**Dataset & Metrics.** Recall that Unicorn so far supports seven types of common data matching tasks (see Section 2), including entity matching, entity linking, entity alignment, string matching, column type annotation, schema matching, and ontology matching. We use widely-recognized datasets for evaluation, and the details of datasets and evaluation metrics are listed in Table 1.

**Implementation Details of Unicorn.** For the PLMs, we use the pre-trained base size checkpoints directly on the Hugging Face. For DeBERTa-base models, they use 12 transformer layers and output a 768 dimensional hidden embedding. 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.

**4.2 Evaluation on Unified Prediction**

This section presents the experimental results for unified prediction evaluation (see Section 3). We train a shared model Unicorn via multi-tasking over all training sets and report performance on various test sets. Due to the space limit, we omit the empirical study of evaluating different PLMs, and only report the result: we find that DeBERTa is the most suitable encoder for Unicorn, which requires structure-aware encoding and high generalization.
Table 1: Dataset Statistics, where |A| (or |B|) is the number of data elements in set A (or B) in each task, \# Matches (\# Non-Matches) is the number of matches (non-matches) in the labeled set D of the task.

| Task Type         | Metric  | Task | |A| | |B| | \# Matches | \# Non-Matches |
|-------------------|---------|------|------|------|----------------|----------------|
| Entity Matching   |         | F1   | Walmart-Amazon (WA) | 2,554 | 22,074 | 962 | 9,280 |
|                   |         |      | DBLP-Scholar (DS)   | 2,616 | 64,263 | 5,347 | 23,360 |
|                   |         |      | Fodor-Zagats (FZ)   | 533   | 331    | 110  | 836   |
|                   |         |      | Amazon (LA)         | 6,907 | 55,923 | 132  | 407   |
|                   |         |      | Beer (Be)           | 3,445 | 3,900  | 68   | 382   |
| Column Type       |         | Acc. | Ethnix (E)          | 620   | 31     | 620  | 18,600 |
| Annotation        |         |      | T2D (T2D)           | 383   | 37     | 383  | 13,788 |
|                   |         |      | Lamaye (Lm)         | 174   | 27     | 179  | 4,519 |
|                   |         |      | T2D (T2D)           | 11,600| 26,025 | 26,666| 131,944|
| String Matching   |         | F1   | Address (Ad)        | 24,600| 29,534 | 9,596| 1,962 |
|                   |         |      | Names (Na)          | 10,341| 15,396 | 5,132| 2,763 |
|                   |         |      | Researchers (Re)    | 8,342 | 43,549 | 4,556| 4,767 |
|                   |         |      | Product (Pr)        | 2,554 | 22,074 | 1,144| 99,410|
|                   |         |      | Citation (Ci)       | 2,616 | 24,623 | 5,347| 34,132|
| Schema Matching   |         | F1   | Fabricated Datasets (Fa) | 11,172 | 11,352 | 7,092| 109,762|
|                   |         |      | DeepMDatasets (DM)  | 41    | 41     | 41   | 268   |
| Ontology Matching |         | Acc. | Cornell-Washington (CW) | 176   | 166    | 53   | 285   |
|                   |         |      | SNRPS 15K (SYG)     | 15,000| 15,000 | 15,000| 38,891|
|                   |         |      | SNRPS 15K (SYG)     | 15,000| 15,000 | 15,000| 38,492|

Figure 5: 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.

Exp-1: Which strategy performs well in Mixture-of-Experts of Unicorn?

Table 2 summarizes the results for all tasks. We compare three variants: Unicorn w/o MoE (only an Encoder and a Matcher), standard Unicorn (with an Encoder, a MoE layer, and a Matcher), and Unicorn++ (an optimized Unicorn with MoE for Expert routing, see Section 3.3.2).

The experimental results show Unicorn and Unicorn++ outperform Unicorn w/o MoE, highlighting the effectiveness of the MoE layer. Specifically, Unicorn++ yields the best results across most tasks, demonstrating the benefits of Expert routing. We analyze the Experts' average utilization weights using heatmaps in Figure 5. For Unicorn, all tasks predominantly utilize the first four Experts with evenly distributed weights (Figure 5 (a)). In contrast, Unicorn++ shows a clearer distinction among Experts (Figure 5 (b)), with similar tasks aggregating to specific experts (e.g., EA tasks to Expert 1, CTA tasks to Expert 2). This indicates the efficacy of our optimization, which includes additional loss functions in Equation (10).

Finding 1: The MoE intermediate layer is helpful for Unicorn, while our MoE optimization for Expert routing further improves the overall performance.

Exp-2: 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 in Table 2, where previous SOTA shows the results of the best task-specific model, which are reported by the existing papers, for each corresponding task.

The experimental results show that our unified models (i.e., Unicorn and Unicorn++) outperform the previous SOTA methods on 15 over 20 tasks. On average, Unicorn++ achieves a score of 94.56, compared to the previous SOTA’s 91.84. Moreover, another advantage of our approach is the significantly reduced model size due to task unification: 147M for Unicorn (comprising 139M for DeBERTa-base and 8M for MoE layer) versus 995.5M for previous SOTA methods, where we compute the number of parameters of models to measure the model size.

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.

Finding 2: 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.

4.3 Evaluation on Zero-Shot Prediction

Exp-3: How does Unicorn perform on unseen tasks with a zero-shot setting?

We evaluate the zero-shot capability of our unified model Unicorn. 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 3, the results of Unicorn with zero label are comparable with previous SOTA methods with many labels.

Finding 3: Unicorn can be effectively applied to new tasks in a zero-shot setting such that pairs in the new task are unlabeled.

Exp-4: Whether our proposed MoE and instruction techniques are helpful in the zero-shot setting?
Table 2: 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Task</th>
<th>Metric</th>
<th>Unicorn w/o MoE</th>
<th>Unicorn</th>
<th>Unicorn++</th>
<th>Previous SOTA (Paper)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM</td>
<td>Walmart-Amazon</td>
<td>F1</td>
<td>85.12</td>
<td>86.89</td>
<td>90.93</td>
<td>86.76 (Ditto [24])</td>
</tr>
<tr>
<td></td>
<td>DBLP-Scholar</td>
<td>F1</td>
<td>95.38</td>
<td>95.64</td>
<td>96.22</td>
<td>95.07 (Ditto [24])</td>
</tr>
<tr>
<td></td>
<td>Fodor-Zagats</td>
<td>F1</td>
<td>97.75</td>
<td>100</td>
<td>97.47</td>
<td>100 (Ditto [24])</td>
</tr>
<tr>
<td></td>
<td>Tunes-Amazons</td>
<td>F1</td>
<td>94.74</td>
<td>96.45</td>
<td>98.18</td>
<td>97.96 (Ditto [24])</td>
</tr>
<tr>
<td></td>
<td>Beer</td>
<td>F1</td>
<td>90.32</td>
<td>90.32</td>
<td>87.5</td>
<td>91.37 (Ditto [24])</td>
</tr>
<tr>
<td>CTA</td>
<td>Effthymiou</td>
<td>Acc.</td>
<td>98.08</td>
<td>98.42</td>
<td>98.44</td>
<td>90.4 (TURL [8])</td>
</tr>
<tr>
<td></td>
<td>T2D</td>
<td>Acc.</td>
<td>98.81</td>
<td>99.11</td>
<td>99.21</td>
<td>96.6 (HNN+P2Vec [2])</td>
</tr>
<tr>
<td></td>
<td>Limaye</td>
<td>Acc.</td>
<td>96.11</td>
<td>96.75</td>
<td>97.32</td>
<td>96.8 (HNN+P2Vec [2])</td>
</tr>
<tr>
<td>EL</td>
<td>T2D</td>
<td>F1</td>
<td>79.96</td>
<td>91.96</td>
<td>92.25</td>
<td>85 (Hybrid I [15])</td>
</tr>
<tr>
<td></td>
<td>Limaye</td>
<td>F1</td>
<td>83.12</td>
<td>86.78</td>
<td>87.9</td>
<td>82 (Hybrid II [15])</td>
</tr>
<tr>
<td>StM</td>
<td>Address</td>
<td>F1</td>
<td>97.81</td>
<td>98.68</td>
<td>99.47</td>
<td>99.91 (Falcon [30])</td>
</tr>
<tr>
<td></td>
<td>Names</td>
<td>F1</td>
<td>86.12</td>
<td>91.19</td>
<td>96.8</td>
<td>95.72 (Falcon [30])</td>
</tr>
<tr>
<td></td>
<td>Researchers</td>
<td>F1</td>
<td>96.59</td>
<td>97.66</td>
<td>97.93</td>
<td>97.81 (Falcon [30])</td>
</tr>
<tr>
<td></td>
<td>Product</td>
<td>F1</td>
<td>84.61</td>
<td>82.9</td>
<td>86.06</td>
<td>67.18 (Falcon [30])</td>
</tr>
<tr>
<td></td>
<td>Citation</td>
<td>F1</td>
<td>96.34</td>
<td>96.27</td>
<td>96.64</td>
<td>90.99 (Falcon [30])</td>
</tr>
<tr>
<td>ScM</td>
<td>FabricatedDatasets</td>
<td>Recall</td>
<td>81.19</td>
<td>89.6</td>
<td>89.35</td>
<td>81 (Valentine [21])</td>
</tr>
<tr>
<td></td>
<td>DeepMDatasets</td>
<td>Recall</td>
<td>66.67</td>
<td>96.3</td>
<td>96.3</td>
<td>100 (Valentine [21])</td>
</tr>
<tr>
<td>OM</td>
<td>Cornell-Washington</td>
<td>Acc.</td>
<td>90.64</td>
<td>92.34</td>
<td>90.21</td>
<td>80 (GLUE [11])</td>
</tr>
<tr>
<td>EA</td>
<td>SRPRS: DBP-YG</td>
<td>Hits@1</td>
<td>99.46</td>
<td>99.67</td>
<td>99.49</td>
<td>100 (BERT-INT [36])</td>
</tr>
<tr>
<td></td>
<td>SRPRS: DBP-WD</td>
<td>Hits@1</td>
<td>97.11</td>
<td>97.22</td>
<td>97.28</td>
<td>99.6 (BERT-INT [36])</td>
</tr>
<tr>
<td></td>
<td><strong>AVG</strong></td>
<td></td>
<td><strong>90.6</strong></td>
<td><strong>94.21</strong></td>
<td><strong>94.56</strong></td>
<td><strong>91.84</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Model Size</strong></td>
<td></td>
<td><strong>139M</strong></td>
<td><strong>147M</strong></td>
<td><strong>147M</strong></td>
<td><strong>995.5M</strong></td>
</tr>
</tbody>
</table>

Table 3: Zero-shot Performance of Unicorn (# of labels is the number of labels needed by SOTA methods). Unicorn-ins is with the instruction technique.

<table>
<thead>
<tr>
<th>Type</th>
<th>Task</th>
<th>Metric</th>
<th>Unicorn w/o MoE</th>
<th>Unicorn</th>
<th>Unicorn-ins</th>
<th>SOTA (# of labels)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM</td>
<td>DS</td>
<td>F1</td>
<td>90.91</td>
<td>95.39</td>
<td>97.08</td>
<td>95.6 (22.965)</td>
</tr>
<tr>
<td>CTA</td>
<td>Lim</td>
<td>Acc.</td>
<td>96.2</td>
<td>96.59</td>
<td>96.5</td>
<td>96.8 (80)</td>
</tr>
<tr>
<td>EL</td>
<td>Lim</td>
<td>F1</td>
<td>74.16</td>
<td>78.92</td>
<td>82.8</td>
<td>82 (-)</td>
</tr>
<tr>
<td>StM</td>
<td>Pr</td>
<td>F1</td>
<td>60.71</td>
<td>74.92</td>
<td>78.76</td>
<td>67.18 (1,020)</td>
</tr>
<tr>
<td>ScM</td>
<td>DM</td>
<td>Recall</td>
<td>74.07</td>
<td>92.59</td>
<td>96.3</td>
<td>100 (-)</td>
</tr>
<tr>
<td>EA</td>
<td>SWD</td>
<td>Hits@1</td>
<td>95.55</td>
<td>97.25</td>
<td>96.17</td>
<td>99.6 (4,500)</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
<td></td>
<td></td>
<td><strong>81.93</strong></td>
<td><strong>89.28</strong></td>
<td><strong>91.27</strong></td>
<td><strong>90.2</strong></td>
</tr>
</tbody>
</table>

We evaluate the performance of zero-shot learning on new tasks for three Unicorn variants: Unicorn w/o MoE, Unicorn, and Unicorn-ins (with zero-shot instruction as presented in Section 3.2.1). The results in Table 3 reveal that Unicorn w/o MoE underperforms compared to the other variants, highlighting the MoE Layer’s role in leveraging knowledge from seen tasks for new unseen tasks. The MoE layer effectively aligns distributions between new and existing tasks, facilitating adaptation to new tasks.

Unicorn-ins shows the best performance, surpassing standard Unicorn by approximately 2% on average (Table 3). Recent studies like Wang et al. [39, 41, 28] explore advanced instruction techniques for such tasks, which we aim to investigate in future work.

Finding 4: There is great potential for studying Mixtures-of-Experts and instruction on Unicorn to improve the performance of zero-shot predictions.

Exp-5: How does Unicorn perform for unseen tasks of totally new task types?

We evaluate the performance of Unicorn for unseen tasks of totally new task types. To this end, we conduct experiments with Unicorn-ins, a version of Unicorn improved by our proposed instruction technique. The results are reported in Table 4, where each row represents an unseen task of a total new task type. Take the first row of Table 4 as an example. For testing DBLP-Scholar of entity matching, we remove all the tasks/datasets of entity matching from the training data of Unicorn-ins. We can see that the performance of Unicorn-ins is comparable to that of the SOTA specific models, e.g., 88.41 vs. 90.2 on average, and some-
times Unicorn-ins is even better. For example, on the Product task, Unicorn-ins with zero labels performs better than the SOTA specific model with 1,020 labels (74.26 vs. 67.18).

The results show that our unified model is competitive to SOTA specific models even if it has never seen the type of a task and has no specific labels from the task.

The results inspire us to conduct a case study on the knowledge sharing capability of Unicorn across different task types. We analyze how the performance of one task type is affected when trained with or without another task type.

(1) Similar Task Types. We explore task types with the same data elements, such as column type annotation (CTA) and schema matching (ScM), both involving the column element. Figure 6 (a) indicates that training Unicorn with CTA tasks significantly benefits ScM tasks (e.g., DeepM-Datasets, 96.3 vs. 70.37 on Recall). Similarly, Figure 6 (b) shows ScM tasks enhance CTA task performance (e.g., Limayev, 95.96 vs. 85.89 on Acc.).

(2) Similar Tasks. We also examine tasks in similar domains, such as products or citations. For instance, entity matching (EM) and string matching (StM) tasks in the product domain show knowledge sharing benefits. Figure 6 (c) shows that training Unicorn with EM tasks is helpful for improving the performance of the StM task Product (59.98 vs. 18.76 on F1), and vice versa (see Figure 6 (d)).

Finding 5: Unicorn performs well for unseen tasks of totally new task types in the zero-shot setting. Moreover, Unicorn enables the possible opportunities of knowledge sharing among different task types.

5. RELATED WORK

Data Matching Tasks. The “data matching” process is central to most, if not all, data integration problems. Existing approaches (e.g., Ditto [24], TURL [8], BERT-INT [36] and so on) 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). Different from them, Unicorn aims at a unified matching model that can be used for multiple matching tasks and datasets.

Transformer-based Language Models. The Transformer architecture, introduced in “Attention Is All You Need” [38], initially aimed at natural language processing, has been successfully adapted for image processing [12] and multimodality tasks like DeepMind’s Gato [34]. Furthermore, Transformer-based models such as BERT [9] and RoBERTa [25] have been applied in database tasks, including Text-to-SQL translation (PASTA [18], SCPrompt [17]) and data lake querying (Symphony [3]), as well as data integration tasks like entity matching (Ditto [24]) and blocking in entity matching (DeepBlocker [37]). Motivated by these advancements, this paper explores the potential of a unified Transformer-based model for various matching tasks essential in data integration and management.

Mixture of Experts Models. Mixture-of-Experts abbreviated as MoE and referenced in [20, 35, 16, 22], is an ensemble technique where multiple experts focus on different sub-tasks. Each expert in MoE specializes in a segment of the task space, with a gating network integrating their outputs. Widely adopted by major tech companies, MoE has shown its effectiveness in diverse applications.

In the context of Unicorn, MoE offers significant advantages. Primarily, Unicorn, designed to handle various data matching tasks, benefits from the specialized focus of each expert in MoE on distinct data types and semantics. Secondly, MoE enhances the scalability and adaptability of Unicorn, allowing for easy extension to novel matching tasks, such as the tuple-to-image matching discussed in Section 6.

6. CONCLUSION AND FUTURE WORK

We have proposed Unicorn, a 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 developed a general framework for Unicorn that employs an Encoder, a Mixture-of-Experts and a Matcher. We conducted 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.

We further discuss the future impact and developments that this research may trigger. Firstly, the optimized MoE and instruction we discussed lead to obvious improvements. Thus, it is worthwhile to explore more optimization techniques in the unified model, e.g., using data augmentation to synthesize new labeled data, to reduce human labeling cost and improve model performance. Secondly, 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. Thirdly, recent progress on large language models (LLMs), such as PaLM [4], ChatGPT and GPT-4 [29], has enabled the possibilities of extending Unicorn to support all tasks in data preparation, such as error detection, missing value imputation, etc. Despite some very recent attempts [23, 43], it still calls for a unified system that has strong generalizability to support unseen tasks.

7. ACKNOWLEDGEMENT

This work was partly supported by the NSF of China (62122090, 62072461, U1911203, 62072458, 62232009 and 61925205), National Key Research and Development Program of China (2020YFB2104101), the Beijing Natural Science Foundation (L222006), the Research Funds of Renmin University of China, Huawei, TAL education, and Beijing National Research Center for Information Science and Technology (BNRist).
8. REFERENCES


