Unicorn: A Unified Multi-Tasking Matching Model

Ju Fan[1], Jianhong Tu[1], Guoliang Li[2], Peng Wang[1], Xiaoyong Du[1]
Xiaofeng Jia[3], Song Gao[3], Nan Tang[4]*
[1]Renmin University of China, Beijing, China [2]Tsinghua University, Beijing, China
[3]Beijing Big Data Centre, Beijing, China [4]HKUST (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 (*e.g.,* TURL [8]) cannot be shared with an entity matching model (*e.g.,* 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.

*Corresponding author.

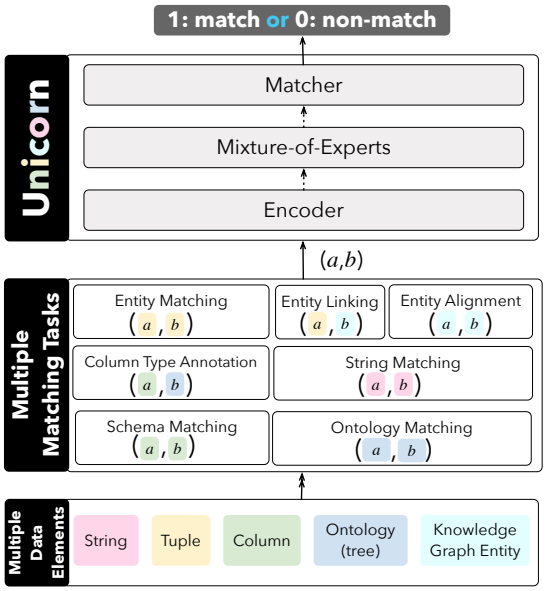


Figure 1: `Unicorn` is a unified model for "data matching" task in data integration. So far, it supports seven matching tasks in data integration for a pair (a, b) where a or b may be of the same or different types of data elements.

To address the challenges, we develop a general framework consisting of three key modules: an `Encoder`, a `Mixture-of-Experts` layer, and a `Matcher` (see Figure 1). The `Encoder` converts any pair of elements (a, b) with heterogeneous formats into a learned representation. Specifically, it serializes any pair of elements into text while still preserving their inherent structure, and employs a unified pre-trained language model for effective encoding. The `Mixture-of-Experts` layer enhances the learned representation into a better representation, in order to align matching semantics of various tasks or datasets. The `Matcher` predicts either 1 (for match) or 0 (for non-match) by taking the above representation as input.

Contributions. We make the following contributions.

(1) As far as we know, `Unicorn` is the first unified multi-tasking matching model for data integration. We formally define the problem of data matching tasks and introduce an overview of the `Unicorn` framework.

(2) We develop effective techniques for two key modules in the `Unicorn` framework, *i.e.,* a unified representation learning method for the `Encoder` module, and effective methods for the `Mixture-of-Experts` layer.

(3) Extensive experiments using 20 datasets for the 7 common data matching tasks show that `Unicorn`, as a unified model, outperforms the state-of-the-art specific models on most tasks and on average.

(4) We make a unified benchmark for multiple data matching tasks available at Github. We publish the source code of `Unicorn`, and provide the trained `Unicorn` model with multi-task learning in Hugging Face, so that researchers and practitioners can either use it out-of-the-box, or fine-tune it for specific tasks.

`https://github.com/ruc-datalab/Unicorn`
`https://huggingface.co/RUC-DataLab/unicorn-plus-v1`

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 $\langle \texttt{word}_i \rangle_{1 \le i \le k}$.

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

Column. A *column* is a set of values $\{\texttt{val}_i\}_{1 \le i \le k}$ of a particular attribute `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 $\texttt{ont} = \{\texttt{node}_i, \texttt{pnode}_i\}_{1 \le i \le k}$, where $\texttt{pnode}_i$ is the parent of $\texttt{node}_i$. 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 $\texttt{KG} = (E, R, A, V, T_r, P_a)$, where $\texttt{ent} \in E$, $\texttt{rel} \in R$, $\texttt{attr} \in A$, and $\texttt{val} \in V$ represent an entity, a relation, an attribute, and an attribute value respectively. Moreover, $(\texttt{ent}_i, \texttt{rel}_k, \texttt{ent}_j) \in T_r(\texttt{ent}_i)$ denotes a relational triple, and $(\texttt{attr}_k, \texttt{val}_j) \in P_a(\texttt{ent}_i)$ denotes an attribute-value pair.

2.2 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 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 `KG-Entity` in a knowledge base. 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.

(6) **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.

(7) **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 adaption 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 $\hat{y}$ 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 $\mathcal{T} = \{T_i\}$. 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 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 2 (a) shows three example data matching tasks, *i.e., 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 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 $\mathbf{x}$, *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. 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). The basic idea is to transform original features of the pairs into an aligned feature space, *i.e.,* $\mathbf{x}' = \mathsf{MoE}(\mathbf{x})$. See Section 3.3 for details.

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 [14] and Ditto [24]), which is denoted as $\hat{y} = \mathsf{M}(\mathbf{x}')$.

Multi-task Training. We adopt multi-task supervised learning [44, 6] to train `Unicorn`, using 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 $\mathcal{D}_i^{\mathsf{tst}}$ of **any** task $T_i \in \mathcal{T}$. Note that $\mathcal{D}_i^{\mathsf{tst}}$ is *unlabeled* and disjoint with the training set $\mathcal{D}_i$ of T_i.

(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 [31, 40] such that the pairs in the new task have **zero** labels.

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:

$$x = \mathsf{S}(a, b) = \texttt{[CLS]}\ \mathsf{S}(a)\texttt{[SEP]}\ \mathsf{S}(b)\ \texttt{[SEP]} \tag{2}$$

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.

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 [9], to serialize `str` into a sequence of sub-words (tokens) as $\mathsf{S}(\texttt{str}) = \texttt{token}_1\ \texttt{token}_2\ \ldots\ \texttt{token}_{k'}$.

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.

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

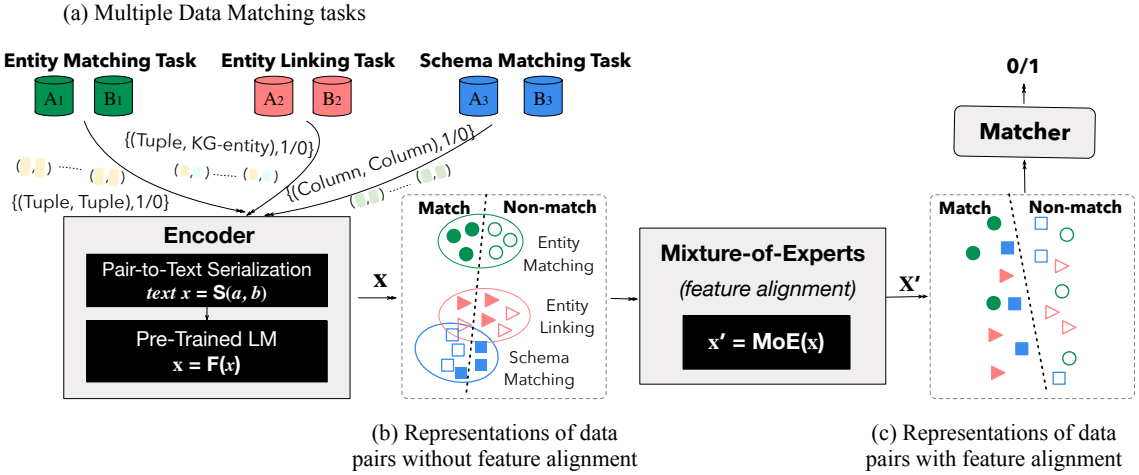


Figure 2: An overview of our proposed `Unicorn` framework. (a) `Unicorn` learns from multiple data matching tasks or datasets. (b) A generic `Encoder` converts any pair of data elements (a, b) into a representation in the form of a high-dimensional vector. (c) A `Mixture-of-Experts` layer aligns matching 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.

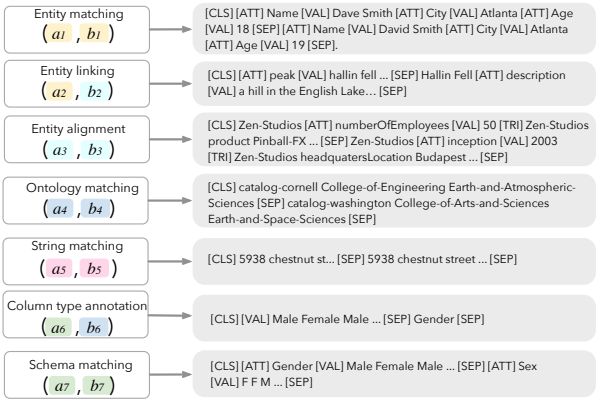


Figure 3: A generic pair-to-text serialization that serializes pairs from data matching tasks into text sequences.

as $\mathsf{S}(\texttt{col}) = \texttt{[ATT] attr [VAL] val}_1\ \texttt{val}_2 \ldots\ \texttt{val}_k$. Note that, in the case of too many values in the column, we randomly select a proportion of the values.

Ontology serialization. Given a tree-based ontology `ont` and a specific $\texttt{node}_k$ in the ontology, we represent it as a sequence by concatenating all nodes in the path from root to $\texttt{node}_k$ as $\mathsf{S}(\texttt{node}_k) = \texttt{node}_1\ \texttt{node}_2 \ldots\ \texttt{node}_k$.

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 $\{(\texttt{attr}_i, \texttt{val}_i)\}_{1 \le i \le k}$, but also some relational triples with other entities $\{\langle \texttt{sub}, \texttt{rel}_i, \texttt{obj}_i \rangle\}_{1 \le i \le m}$. Based on the structure, we serialize the KG-entity `sub` into sequence

$$\mathsf{S}(\texttt{sub}) = \texttt{sub [ATT] attr}_1\ \texttt{[VAL] val}_1 \ldots \texttt{[ATT] attr}_k\ \texttt{[VAL] val}_k\ \texttt{[TRI] sub rel}_1\ \texttt{obj}_1 \ldots \texttt{[TRI] sub rel}_m\ \texttt{obj}_m,$$

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 $\mathsf{S}(a_3, b_3)$="`[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:

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

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, $\mathsf{MoE}(\mathbf{x}) : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ transforms an original feature vector $\mathbf{x} \in \mathbb{R}^{d_1}$ to a new vector $\mathbf{x}' \in \mathbb{R}^{d_2}$. The common

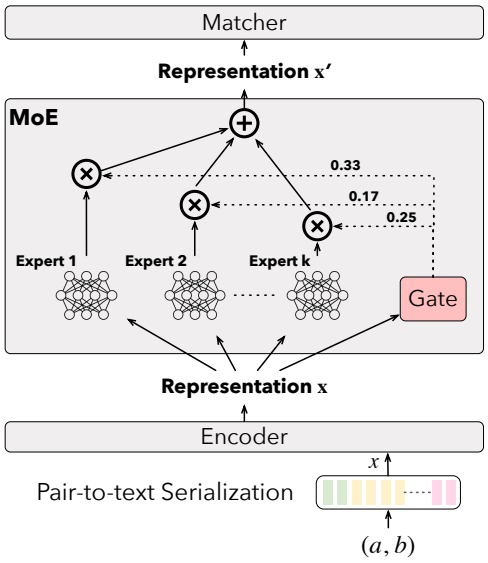


Figure 4: An overview of the `Mixture-of-Experts` layer. Experts are a set of neural networks to map $\mathbf{x}$ to different representations, and Gating is to combine the outputs of Experts according to learned weights based on input $\mathbf{x}$.

`Mixture-of-Experts` architecture [20] consists of Experts and Gating. Experts are neural networks mapping $\mathbf{x}$ to $\mathbf{x}'$, while Gating combines these outputs based on $\mathbf{x}$.

The neural network design and training algorithm for MoE 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_i 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):

$$\mathbf{x}_i = \texttt{LeakyReLU}(\mathbf{x}W^{(i)}) \tag{4}$$

where $W^{(i)} \in \mathbb{R}^{d_1 \times d_2}$ 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):

$$\mathbf{g} = \texttt{Softmax}\big(\texttt{LeakyReLU}(\mathbf{x}W_1^{\texttt{G}})W_2^{\texttt{G}}\big) \tag{5}$$

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 (4) and (5), 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 $\mathbf{x}'$,

$$\mathbf{x}' = \text{MoE}(\mathbf{x}) = g_1 \cdot \mathbf{x}_1 + g_2 \cdot \mathbf{x}_2 + \cdots + g_k \cdot \mathbf{x}_k. \tag{6}$$

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 $\hat{y} = \text{M}(\mathbf{x}')$. Loss is computed using cross entropy function $\mathcal{L}_{\text{CE}}$ 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* $\mathcal{L}_{\texttt{Bal}}$, which is the squared coefficient of variation of $\mathbf{u}$:

$$\mathbf{u} = (\sum_{\mathbf{x}\in\mathcal{D}} g_1, \sum_{\mathbf{x}\in\mathcal{D}} g_2, \ldots, \sum_{\mathbf{x}\in\mathcal{D}} g_k), \tag{7}$$

where $\sum_{\mathbf{x}\in\mathcal{D}} g_i$ is the sum of gating weights for Expert_i on all the training examples in $\mathcal{D}$.

$$\mathcal{L}_{\texttt{Bal}} = [\frac{\sigma(\mathbf{u})}{\mu(\mathbf{u})}]^2 \tag{8}$$

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 $\mathcal{L}_{\texttt{Ent}}$:

$$\mathcal{L}_{\texttt{Ent}} = \mathbb{E}_{(\mathbf{x}',y)} - \sum_{i=1}^{k} g_i \cdot \log(g_i) \tag{9}$$

The new loss function for training `Unicorn` is:

$$\mathcal{L}_{\texttt{new}} = \mathcal{L} + \mathcal{L}_{\texttt{Bal}} + \mathcal{L}_{\texttt{Ent}} \tag{10}$$

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.

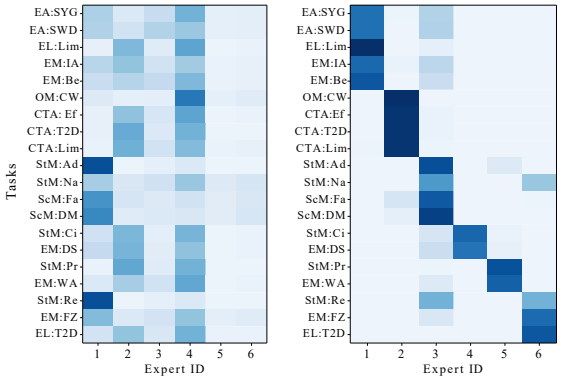
Implementation Details of `Unicorn`. For the PLMs, we use the pre-trained base size checkpoints directly on the Hugging Face. For DeBERTa-base models, they use 12 transformer layers and output a 768 dimensional hidden embedding. For the MoE layer, we choose expert number from 2 to 15, and set hidden 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 $\mathcal{D}$ of the task.**

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



(a) `Unicorn` (b) `Unicorn++`

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.**

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 [24])
	DBLP-Scholar	F1	95.38	95.64	**96.22**	95.6 (Ditto [24])
	Fodors-Zagats	F1	97.78	**100**	97.67	**100** (Ditto [24])
	iTunes-Amazon	F1	94.74	96.43	**98.18**	97.06 (Ditto [24])
	Beer	F1	90.32	90.32	87.5	**94.37** (Ditto [24])
CTA	Efthymiou	Acc.	98.08	98.42	**98.44**	90.4 (TURL [8])
	T2D	Acc.	98.81	99.14	**99.21**	96.6 (HNN+P2Vec [2])
	Limaye	Acc.	96.11	96.75	**97.32**	96.8 (HNN+P2Vec [2])
EL	T2D	F1	79.96	91.96	**92.25**	85 (Hybrid I [15])
	Limaye	F1	83.12	86.78	**87.9**	82 (Hybrid II [15])
StM	Address	F1	97.81	98.68	99.47	**99.91** (Falcon [30])
	Names	F1	86.12	91.19	**96.8**	95.72 (Falcon [30])
	Researchers	F1	96.59	97.66	**97.93**	97.81 (Falcon [30])
	Product	F1	84.61	82.9	**86.06**	67.18 (Falcon [30])
	Citation	F1	96.34	96.27	**96.64**	90.98 (Falcon [30])
ScM	FabricatedDatasets	Recall	81.19	**89.6**	89.35	81 (Valentine [21])
	DeepMDatasets	Recall	66.67	96.3	96.3	**100** (Valentine [21])
OM	Cornell-Washington	Acc.	90.64	**92.34**	90.21	80 (GLUE [11])
EA	SRPRS: DBP-YG	Hits@1	99.46	99.67	99.49	**100** (BERT-INT [36])
	SRPRS: DBP-WD	Hits@1	97.11	97.22	97.28	**99.6** (BERT-INT [36])
AVG			90.8	94.21	**94.56**	91.84
Model Size			139M	147M	147M	995.5M

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.**

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

Table 4: **Zero-shot Performance of** `Unicorn` **for new unseen task types. (# of labels is the number of labels needed by SOTA methods).**

Type	Task	Metric	Unicorn-ins	SOTA (# of labels)
EM	DS	F1	94.5	**95.6** (22,965)
CTA	Lim	Acc.	96.23	**96.8** (80)
EL	Lim	F1	79.59	**82** (-)
StM	Pr	F1	**74.26**	67.18 (1,020)
ScM	DM	Recall	88.89	**100** (-)
EA	SWD	Hits@1	97	**99.6** (4,500)
AVG			88.41	**90.2**

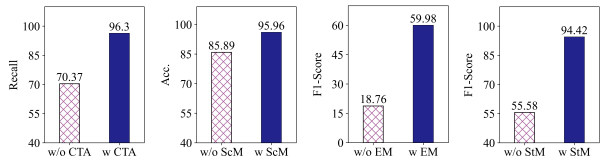
We evaluate the performance of zero-shot learning on new tasks for three `Unicorn` variants: `Unicorn` w/o MoE, `Unicorn`, and `Unicorn`-ins (with zero-shot instruction as presented in Section 3.2.1). The results in Table 3 reveal that `Unicorn` w/o MoE underperforms compared to the other variants, highlighting 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 `Mixture-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-



(a) ScM:DM (b) CTA:Lim (c) StM:Pr (d) EM:DS

Figure 6: Knowledge sharing capability of `Unicorn` across different task types. (a) and (b) evaluate knowledge sharing between schema matching (ScM) and column type annotation (CTA), while (c) and (d) consider entity matching (EM) and string matching (StM).

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., Limaye, 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], SCPromt [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

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. Sutton. Learning semantic annotations for tabular data. *arXiv preprint arXiv:1906.00781*, 2019.

[3] Z. Chen, Z. Gu, L. Cao, J. Fan, S. Madden, and N. Tang. Symphony: Towards natural language query answering over multi-modal data lakes. In *2023 Conference on Innovative Data Systems Research (CIDR)*, 2023.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

[5] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, and K. Stefanidis. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)*, 53(6):1–42, 2020.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[7] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[8] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. TURL: table understanding through representation learning. *Proceedings of the VLDB Endowment*, 14(3):307–319, 2020.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.

[11] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *The VLDB journal*, 12(4):303–319, 2003.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[13] H. L. Dunn. Record linkage. *American Journal of Public Health*, 36(12):1412–1416, 1946.

[14] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.

[15] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *International Semantic Web Conference*, pages 260–277. Springer, 2017.

[16] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

[17] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du. Few-shot text-to-sql translation using structure and content prompt learning. *Proceedings of the ACM on Management of Data*, 1(2):1–28, 2023.

[18] Z. Gu, J. Fan, N. Tang, P. Nakov, X. Zhao, and X. Du. 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*, pages 4971–4983. Association for Computational Linguistics, 2022.

[19] P. He, X. Liu, J. Gao, and W. Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[20] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[21] C. Koutras, G. Siachamis, A. Ionescu, K. Psarakis, J. Brons, M. Fragkoulis, C. Lofi, A. Bonifati, and A. Katsifodimos. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 468–479. IEEE, 2021.

[22] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[23] P. Li, Y. He, D. Yashar, W. Cui, S. Ge, H. Zhang, D. R. Fainman, D. Zhang, and S. Chaudhuri. Table-gpt: Table-tuned GPT for diverse table tasks. *CoRR*, abs/2310.09263, 2023.

[24] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14(1):50–60, 2020.

[25] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[26] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H.

Chi. 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*, pages 1930–1939, 2018.

[27] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.

[28] A. Narayan, I. Chami, L. J. Orr, and C. Ré. Can foundation models wrangle your data? *CoRR*, abs/2205.09911, 2022.

[29] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[30] G. Paul Suganthan, A. Ardalan, A. Doan, and A. Akella. Smurf: Self-service string matching using random forests. *Proceedings of the VLDB Endowment*, 12(3), 2019.

[31] F. Pourpanah, M. Abdar, Y. Luo, X. Zhou, R. Wang, C. P. Lim, X.-Z. Wang, and Q. J. Wu. A review of generalized zero-shot learning methods. *IEEE transactions on pattern analysis and machine intelligence*, 2022.

[32] Z. Qin, Y. Cheng, Z. Zhao, Z. Chen, D. Metzler, and J. Qin. Multitask mixture of sequential experts for user activity streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3083–3091, 2020.

[33] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[34] S. E. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. A generalist agent. *CoRR*, abs/2205.06175, 2022.

[35] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[36] X. Tang, J. Zhang, B. Chen, Y. Yang, H. Chen, and C. Li. 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*, pages 3174–3180, 2021.

[37] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. Fung, and A. Doan. Deep learning for blocking in entity matching: a design space exploration. *Proceedings of the VLDB Endowment*, 14(11):2459–2472, 2021.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] P. Wang, A. Yang, R. Men, J. Lin, S. Bai, Z. Li, J. Ma, C. Zhou, J. Zhou, and H. Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*, pages 23318–23340. PMLR, 2022.

[40] W. Wang, V. W. Zheng, H. Yu, and C. Miao. A survey of zero-shot learning: Settings, methods, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–37, 2019.

[41] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Naik, A. Ashok, A. S. Dhanasekaran, A. Arunkumar, D. Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109, 2022.

[42] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Ç. Demiralp, and W. Tan. Sato: Contextual semantic type detection in tables. *Proceedings of the VLDB Endowment*, 13(11):1835–1848, 2020.

[43] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada. Jellyfish: A large language model for data preprocessing. *CoRR*, abs/2312.01678, 2023.

[44] Y. Zhang and Q. Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.