Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning

Chen Zhao*
University of Maryland, College Park
chenz@cs.umd.edu

Yeye He
Microsoft Research, Redmond
yeyehe@microsoft.com

ABSTRACT

Entity matching (EM), also known as entity resolution, fuzzy join, and record linkage, refers to the process of identifying records corresponding to the same real-world entities from different data sources. It is an important and long-standing problem in data integration and data mining. So far progresses have been made mainly in the form of model improvements, where models with better accuracy are developed when large amounts of training data is available. In real-world applications we find that advanced approaches can often require too many labeled examples that is expensive to obtain, which has become a key obstacle to wider adoption.

We in this work take a different tack, proposing a transfer-learning approach to EM, leveraging pre-trained EM models from large-scale, production knowledge bases (KB). Specifically, for each entity-type in KB, (e.g., location, organization, people, etc.), we use rich synonymous names of known entities in the KB as training data, to pre-train type-detection and EM models for each type, using a novel hierarchical neural network architecture we develop. Given a new EM task, with little or no training data, we can either fine-tune or directly leverage pre-trained EM models, to build end-to-end, high-quality EM systems. Experiments on a variety of real EM tasks suggest that the pre-trained approach is effective and outperforms existing EM methods.[1].

ACM Reference Format:
Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313578

1 INTRODUCTION

Entity matching (EM), also known as entity resolution, fuzzy join, and record linkage, has numerous important applications such as knowledge fusion [29, 51], customer record deduplication [22, 61], etc. EM has been a long-standing problem in the data mining and data integration community, where extensive work has resulted in a long and fruitful line of research (e.g., see surveys in [28, 30, 45]).

*Work done at Microsoft Research.
[1]We plan to release the pre-trained EM models, and are working through required processes to make this happen. Once approved these models will be released on GitHub at https://github.com/henryzhao5852/AutoEM.

A typical EM task in a relational setting, is to predict which records from two tables correspond to the same real-world entities. (EM in a graph setting such as knowledge graphs can be cast in a similar manner using graph connections [51]). Figure 1 shows an example of EM task. Given two tables of customer records with information such as customer names and addresses, we need to match records likely corresponding to the same person across the two data sources.

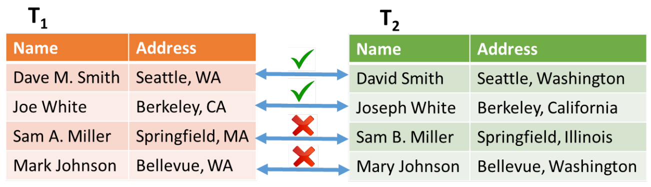
In this example, we can intuitively tell that the first two pairs of records are likely matches despite their differences in string representations – "Joe White" and "Joseph White" likely refer to the same entity, so do "CA" and "California". However minor string differences are not sufficient to ensure matches. To the contrary, there are many record pairs that have minor differences but are clearly non-matches. For example, in the last two pairs of records, "Sam A. Miller" and "Sam B. Miller" are likely not the same person, so are "Mark Johnson" and "Mary Johnson".

As we can see, these match/non-match decisions can be subtle, and are non-trivial to predict accurately. Existing EM approaches such as ML-based ones [10, 15, 61, 65], often require a large amount of training data (labeled match/non-match pairs) *each new EM task*, before accurate EM predictions can be made. It is clearly expensive, and sometimes impossible, to produce a large number of labeled data for *each EM task*, and this has become major obstacles to wider adoption of advanced EM techniques.

EM in real-world business applications. This study is conducted in the context of a commercial CRM (customer relationship management) system, where a pressing need is for enterprises using the CRM system, to match their customer records across data silos in different business applications (CRM, ERP, billing, service, etc.), so that these enterprises can have a unified view of their customers. Customer records are often in relational formats as shown in Figure 1, although the schema of the tables are not always known beforehand (e.g., data can be in non-standard databases or CSV files). A key requirement of EM in such applications, is that matching of customers should ideally work accurately out-of-box with little or no training data from *each enterprise*. EM applications like this make it difficult to apply advanced EM methods from the literature, which typically requires a large amount of labeled data [66].

Transfer learning for EM using pre-trained models. Partly motivated by the real-world EM application, we in this work take a very different tack. We argue that one does not need to re-train EM models from scratch for each new EM task, and we propose a novel transfer-learning approach using pre-trained EM models.

Specifically, our insight is that while each EM task may be different in its own ways (e.g., tables may have different attributes, and attributes have different importance, etc.), the types of attributes involved are often drawn from a set of common attributes (e.g.,



T_1

Name	Address
Dave M. Smith	Seattle, WA
Joe White	Berkeley, CA
Sam A. Miller	Springfield, MA
Mark Johnson	Bellevue, WA

T_2

Name	Address
David Smith	Seattle, Washington
Joseph White	Berkeley, California
Sam B. Miller	Springfield, Illinois
Mary Johnson	Bellevue, Washington

Figure 1: Example entity-matching between two tables.

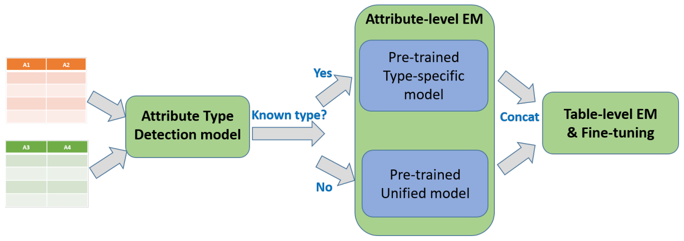

A1
A2
A3
A4
Attribute Type Detection model
Known type?
Yes
No
Attribute-level EM
Pre-trained Type-specific model
Pre-trained Unified model
Concat
Table-level EM & Fine-tuning

Figure 2: System architecture of end-to-end EM.

people-names, addresses, organizations, product-names, etc.). We observe that for each such attribute, the decision of match/non-match *at the attribute-level* can often be pre-trained and determined independent of the overall table-level EM task. For instance, in Figure 1, for the people-names domain, it is rather unambiguous that ("Joe White", "Joseph White") should match, while ("Sam A. Miller", "Sam B. Miller"), or ("Mark Johnson", "Mary Johnson") should not, irrespective of the EM tasks in question.

In addition, we observe that training data for these attribute-level match/non-match decisions are readily available in today's KBs, in the form of "synonymous names" for a large variety of entities (e.g., "Bill Gates" is also known as "William Gates" and "William H. Gates" in KBs). We leverage data harvested from KBs to pre-train accurate *attribute-level EM models* for a variety of domains, using a novel hierarchical deep model architecture we develop, which is shown to address shortcomings of existing methods.

Although the *attribute-level* EM decisions can be pre-trained and made separately, their contribution/importance to the overall *table-level* EM task can vary significantly. For instance, when matching two set of customer-records like in Figure 1, if the address-field of one table is "billing address" and the other is "mailing address", then they are not as important at the table-level (similarly, imagine if one phone-number field is "office-number" while the other is "cell-number"). In this work, we show that using pre-trained attribute-level EM models, and limited training data for each specific table-level EM task, we can quickly converge to accurate table-level EM predictions, through a fine-tuning step leveraging internal representations of these attribute-level EM models.

We would like to note that our pre-training approach to EM coincides with recent trends of employing pre-training models to NLP, including notable recent work such as BERT [26] that are shown to have great success.

Contributions. We make the following contributions.

• We propose a novel, end-to-end architecture for EM, that leverages large-scale and curated KB data, to pre-train models for both attribute type-detection, and attribute-level matches. For each new EM task, these attribute-level signals can be easily combined to produce an aggregate EM decision, using a small amount of training data.

• We develop a new hierarchical deep model to pre-train match/non-match for common types of attributes. This model leverages both character-level and word-level information and is designed to better handle complex attributes.

• We perform extensive experiments using KB data and real table data. Our results show that the proposed approach produces comparable or better results compared to state-of-the-art methods.

2 SYSTEM ARCHITECTURE

The EM problem we consider is simple to state: given two tables T_1 and T_2, with n and m records, respectively, determine for each record in T_1, if it matches with any record in T_2.

Figure 2 shows the end-to-end architecture of the proposed system. At a high level, the online EM prediction system has three main components: (1) Attribute-type detection; (2) Attribute-level EM; and (3) Table-level EM. We discuss each component in turn.

The first component is attribute-type detection, which takes a table as input, and predicts if each attribute/column in the table corresponds to a known KB type T. In the table of Figure 1, for instance, the first column is predicted as the KB type person, the second column as city, etc. These type-detection models are pre-trained offline using rich KB data from a commercial search engine. Specifically, KBs used by Google [5], Microsoft [32] and others have millions of entities for hundreds of common type such as person, city, organization, book, movie, etc. We leverage these (entity $\rightarrow$ type) data to train deep models to detect table column types. This component will be described in Section 4.

The second component is attribute-level EM models and is the central part of our system. It takes as input two entity values (e.g., "Dave M. Smith" and "David Smith" in Figure 1), and produces a score indicating the likelihood of match for the two input. We use two types of attribute-level EM models that are pre-trained offline:
(1) *Type-specific models*: For each known KB type T (e.g. person), we pre-train a separate model to predict match/non-match for values in T. We use synonymous entity names of type T in KB (e.g., "Bill Gates" is also known as "William Gates", "William Henry Gates" and "William H. Gates", etc.) as training data, and develop hierarchical deep models to learn name variations specific to each type T for accurate match/non-match decisions.
(2) *Unified model*: This is a single model that predicts match/non-match for values not in known KB types. While the model architecture is the same as the *type-specific models*, we use synonymous entity names taken from the *union* of many KB types, to pre-train a unified attribute-level EM that captures common name variations across different types (e.g., spelling variations). Such a model is reasonably accurate, and can be fine-tuned using limited training data to quickly fit a new type not known a priori.

As illustrated in Figure 2, at online prediction time the attribute-level EM can take two possible paths using the two types of models above, based on type-detection results. Specifically, if the type of a pair of attributes are detected to be a known KB type T, we apply the type-specific models for T (the upper path), otherwise we apply the general-purpose unified model (the lower path). We will describe this component in Section 3.

The final part of our system is the table-level EM. As discussed earlier, each table-level EM task can be different (e.g., different attributes, and different levels of importance for the same attributes). The table-level EM model starts from pre-trained attribute-level EM, and uses limited training data to quickly converge to aggregate EM decisions. This approach can also leverage pre-trained representations to fine-tune attribute-level EM for types that are not pre-trained, using limited (e.g., a few dozens) training data.

Terminology. Since in this work we will describe data coming from the contexts of relational tables and KBs, sometimes the same concept may be referred to using different names that are more natural in their respective contexts. For instance, "columns" or "attributes" that are more natural in tables, are better described as "entity-types" in KBs; similarly "attribute values" in tables are commonly described as "entity names" in KBs. While we try to keep the names consistent, we will use these names interchangeably in their corresponding contexts when appropriate.

3 ATTRIBUTE-LEVEL ENTITY MATCHING

We start by introducing our attribute-level EM models (in the middle of Figure 2), since they are the central part of the EM system, for which we develop novel hierarchical deep models. We defer the first component on type detection to Section 4, since we use simplified versions of the hierarchical models for type-detection.

Recall that attribute-level EM needs to take two attribute-values as input, and produce a score indicating their likelihood of match, which can be intuitively interpreted as "similarity".

3.1 Training data preparation

From Bing's knowledge graph [3, 32] (which is known as Satori and is similar to Google Knowledge Graph [5]), we select 40 head entity types that are deemed as common and useful for EM tasks (e.g., person, organization, city, book, etc.). In this KB (as well as other popular KBs [51, 56]), each entity e has an attribute called "alias", that lists all alternative/synonymous names of e. For example, the entity "Bill Gates" has alias "William Henry Gates", "William H. Gates", etc. These alternative names are clearly useful to train type-specific attribute-level EM models.

For positive examples, we take pairs of such alternative names, while filtering out pairs with no token overlap. The pairs that are listed as alternative names in KB but with no token overlap are likely semantic synonyms: e.g., "Lady Gaga" is also known as "Stefani Joanne Angelina Germanotta". Such semantic synonyms are too specific that are fine to memorize but difficult to generalize.

We would like to note that similar synonym data are also widely available in a similar manner from other KBs, such as the "also-known-as" relation in Wikidata [69], "alias" relation in Freebase [16], "foaf:nick" relation in DBpedia [12], "means" relation in YAGO [67], "alternateName" relation in Google Knowledge Graph [5]; as well as from standalone entity synonym data feeds [18, 21].

For negative examples, we use pairs of entities (e, e') in KB, whose names have some syntactic similarity. For example, we use "Bill Gates" and "Bill Clinton" as a pair of negative examples, as they resolve to different KB entities, but also share a common token in their names. The reason we require negative pairs to have syntactic similarity is that if the pair are completely different, it is trivial to determine that they should not match (e.g., "Bill Gates" and "Larry Page"). Such pairs would not be as helpful for models to learn. We generate "highly similar" pairs of names that are informative as negative examples as follows: for each entity e, we find top-100 entities in the same type, whose names are most similar to e (similarity to e is first decided based on the number of overlap tokens with e, and then based on Edit distance when there is a tie).

We note that for each canonical entity name from different types, on average we produce 2 to 5 positive examples (synonym names), and exactly 100 negative examples.

Our KB does not currently curates long-form physical mailing addresses (e.g., "206 South Cross Street, Little Rock, AR"), which however are common in EM tasks. In order to complement the KB for address data, we use query logs collected from the "Maps" vertical of the search engine, to obtain variations of addresses (in ways that users would type them), as well as their canonical addresses generated by the search engine. For example, a user query may be "206 South Cross Street, Little Rock, AR", and it is mapped to the canonical address "206 S Cross St, Little Rock, AR 72201" by the search engine. We collect such pairs of addresses as positive examples. And similar to KB types, negative examples are selected from high-similar address pairs that resolve to different canonical addresses. In total, we generate training data for addresses in 9 English-speaking locales (e.g. "en-us", "en-ca", "en-gb", "en-in", etc.), and use these as 9 additional types.

In total we pre-train EM models for these 49 attribute-types. We note that our approach of obtaining training data is general, and can be easily extended. For example, we could add types from KB, or use entity names in other languages from KB as additional types (most entities are curated to have names in many different languages). The same is true for addresses in other languages/locales.

3.2 Hierarchical Model for Attribute-level EM

We observe that positive examples of matching entity names exhibit complex structures and variations in different attribute-types. We make the following observations that motivate us to design a specific model architecture for attribute-level EM.
(1) First, we observe that sub-word/character-level matches are often important: for example, we have ("Dave Smith" = "David Smith"), and ("International Business Machine Corp" = "IBM Corp"), which requires character-level information to be modeled.
(2) In addition, word-level pairs are also an important source of information: for instance, we have ("Bill Gates" = "William Gates"), as well as ("William H. Gates" ≠ "William A. Gates") and ("Mary Miller" ≠ "Mark Miller"), etc. While character-level models are able to capture some of these, for long names with many tokens it can be difficult, such that explicit word-level models would be useful.
(3) Within one input, different words/characters may have different importance. For instance ("IBM Inc." = "IBM Corp."), since in the organization type words like "Inc." and "Corp." are not important; but ("IBM Corp." ≠ "IBS Corp."). The same is true for other types like person. This motivates us to introduce an intra-input, self-attention-like mechanism to learn character/word importance.
(4) Between two input, sometimes the word order may be different, e.g., ("Dave Smith" = "Smith, David"), which calls for an alignment-like, inter-input attention mechanism between the two input strings (reminiscent to attention used in machine-translation [13]).

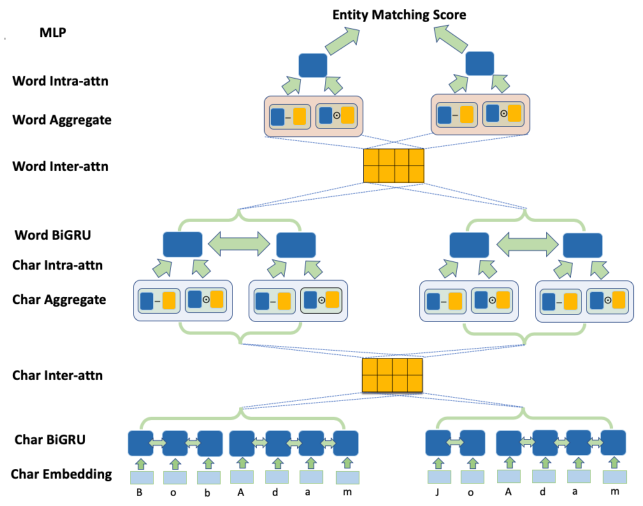


Figure 3: The hierarchical Hi-EM model for attribute-level EM. The two input "Bob Adam" and "Jo Adam" at the bottom pass through a number of layers (GRU, attention, etc.), before producing a match score.

These observations motivate us to develop a hierarchical-EM model shown in Figure 3, which we will refer to as Hi-EM for short. At a high level, the model has a hierarchical structure, which starts with character-level layers (the ones that start with "Char" in Figure 3), but also has upper-layers that explicitly capture word-level information (the layers that start with "Word"). With the hierarchical model, the character-level layers can not only capture fine-grained character-level variations, but also address the common out-of-vocabulary (OOV) issues. At the same time the word-level layers can explicitly leverage word-boundaries (separating characters in different words), which are especially beneficial for long input strings (e.g., data in types like address often have 5-10 words). We note that a similar idea of hierarchical models was recently explored in other contexts [74].

Additionally, for both the character-level and word-level, we introduce layers specifically designed for intra-input attention (within single input), and inter-input attention (between two input strings), which would help the model to learn character/word importance, as well as alignments between two input strings.

We now describe different layers of the model in turn in detail.

3.2.1 *Character-level Layers for Word Representations.*

We will first describe the 5 layers at the bottom of Figure 3. These are at the character-level to ultimately produce word-level representations, and their names all start with "Char". At a high level, we will first encode characters in the input, then look at the other input for alignments using attention, before aggregating to produce word representations for word-level layers.

Character Encoder. This part includes the first two layers: Char-Embedding, and Char-Bi-GRU. In the Char-Embedding layer, given a word $w_i, i \in [i, n]$, with its characters denoted as c_{it}, $t \in [1, l_i]$, we embed the characters to vectors through a character-embedding matrix W_e.

$$e_{it} = W_e * c_{it} \tag{1}$$

Then we pass the embedded vectors e_{it} to a recurrent neural network (RNN) block to obtain contextual information of the characters. In this work we use Bidirectional-Gated-Recurrent-Unit (BiGRU) [13] to capture both forward and backward information (similar to bidirectional LSTM [34]). The resulting character representation is denoted as hc_{it}.

$$hc_{it} = BiGRU(e_{it}) \tag{2}$$

Character Inter-input Attention. For each character representation hc_{it}, we adopt an inter-input attention layer to incorporate the character alignment $hc_j, j \in [1, l]$, where l refers to the length of whole character sequence from the other input. We use a bi-linear function with learned weight W_c to get the attention weights from the character sequence of the other input.

$$\alpha_j = hc_{it} * W_c * hc_j \tag{3}$$

For each character position it, the character information from the other attribute is summarized as

$$a_{it} = \sum_{j=1}^{l} \alpha_j hc_j \tag{4}$$

Character Aggregation and Intra-input Attention. For each character c_{it}, we produce a representation that is the concatenation of the element difference and multiplication between hc_{it} and a_{it}.

$$pc_{it} = [|hc_{it} - a_{it}|; hc_{it} \circ a_{it}] \tag{5}$$

We use the intra-attention layer to re-weight each combined character representation through a linear layer.

$$\beta_{it} = w_c * pc_{it} \tag{6}$$

The final representation for each word r_i is a weighted average of character representation.

$$r_i = \sum_{t=1}^{l_i} \beta_{it} * pc_{it} \tag{7}$$

We obtain word representation of each word from the two input strings, denoted as r_i, $i \in [1, n]$, and r_j, $j \in [1, m]$, where n, m are the total number words from the two input, respectively.

3.2.2 *Word-level Layers for Attribute-value Representations.*

On top of the character-level layers that produce word-level representations, we stack another set of word-level layers for overall attribute-value representations. These layers are designed similarly to include word encoding, inter-input attention, aggregation and finally intra-input attention, before producing a final representation for the full attribute value.

Word Encoder. We first use a BiGRU layer to contextualize each word representation $r_i, i \in [1, n]$.

$$hw_i = BiGRU(r_i) \tag{8}$$

And the same hw_j can be produced for $r_j, j \in [1, m]$.

Word Inter-input Attention. We have another inter-input attention layer to incorporate alignment information with the other input string $hw_j, j \in [1, m]$.

$$\alpha_j = hw_i * W_d * hw_j \tag{9}$$

$$a_i = \sum_{j=1}^{m} \alpha_j hw_j \tag{10}$$

Word Aggregation and Intra-input Attention. We again concatenate the element difference and multiplication of the word representation and the aligned word representation.

$$pw_i = [|hw_i - a_i|; hw_i \circ a_i] \tag{11}$$

Then we apply an intra-attention layer for final attribute-value representation z.

$$\beta_i = w_d * pw_i \tag{12}$$

$$z = \sum_{i=1}^{n} \beta_i * pw_i \tag{13}$$

We denote the final representations of the two input strings so computed as z_p and z_q, respectively.

3.2.3 Final Prediction.

The representation z_p, z_q for a pair of attribute values (P, Q) are concatenated and then pass through a multi-layer perceptron (MLP) layer to produce a final EM score.

$$score(P, Q) = MLP(z_p, z_q) \tag{14}$$

During training, we use logistic regression loss that averages over all N examples as the loss function.

$$loss = \frac{1}{N} \sum_{pos} log(\frac{1}{1 + e^{-score_{pos}}}) + \sum_{neg} log(\frac{1}{1 + e^{score_{neg}}}) \tag{15}$$

3.3 Transfer Learning for EM

For each attribute type T, we train a separate attribute-level EM model that captures the specific characteristics in T (e.g., synonymous tokens, token importance, etc.), which can then make highly accurate match/non-match decisions for data in type T.

However, even though we pre-train attribute-level EM for a large number of types, there will be attributes in EM tasks that are not in the known types. For those attributes we apply transfer-learning as follows, so that even for a new type not known a priori, we could quickly converge to a high-quality EM model.

We take the *union* of data in known attribute types, to build a general-purpose attribute-level EM model, which we will refer to as the *unified-model*. Such a model captures common variations general across many types (e.g., spell variations), and serves as a good starting point to train models for a new attribute-type. With limited training data for the new attribute-type, we take internal representations from the unified-model (right before the MLP layers in Figure 3), and add new MLP layers that can be fine-tuned using new training data to quickly converge to an EM model specific to the new type. Our experiments suggest that this transfer-learning approach produces high-quality results with limited training data.

Finally, for table level EM, we use a similar transfer-learning approach. Based on table attribute types, we use either type-specific attribute-level model or unified-model, to get internal representations every attribute pair (z in Equation (13)). We concatenate all representations, and add an MLP layer at the end for table-level EM. Such a model can be fine tuned end-to-end, using a small amount of table-level training data.

4 ATTRIBUTE TYPE DETECTION

In this section, we describe the first component of our system shown in Figure 2, which is for attribute type detection. Recall that for each value from input table column, we need to detect whether it belongs to known attribute-types T.

4.1 Training data preparation

Our training data used for attribute-type detection is similar to the data for attribute-level EM described in Section 3.1.

We use the same 40 common KB types, and 9 address types for type-detection. Note that each entity in the KB can be associated with one or more types. For example, entity "University of California" is of type "location", "organization", "educational institution", etc.; and entity "Harry Potter" is of type "written book", "film", "person", etc. Since the type hierarchy in the KB is such that types are not mutually exclusive but partially overlapping, and a string name can indeed belong to multiple types, we in this work formulate type-detection as a multi-hot classification problem – given an input string, predict all types it belongs to.

For each type T, we use names of entities in T, or $\{e \in T\}$ as positive examples for training. For negative examples, initially we use entities from $\{e' \notin T\}$. However, this turns out to be problematic, because KB types are often incomplete. For instance, while "University of California" has both types "organization" and "educational institution", another (smaller) university "Gonzaga University" only has type "educational institution" but not "organization" (which it a missing type)[2]. Note that because of the missing type, we may incorrectly use "Gonzaga University" as a negative example for "organization", which confuses the model.

To address this issue, we use a conservative approach to avoid selecting an entity $e \in T_1$ as a negative example for T_2, if its known type T_1 has positive correlation with T_2 (e.g., "organization" and "educational institution"). Specifically, for each pair of types T_1 and T_2, we compute their entity-instance-level point-wise mutual information (PMI) [46], defined as $\frac{|\{e|e \in T_1, e \in T_2\}||\{e \in U\}|}{|\{e \in T_1\}||\{e \in T_2\}|}$, where $\{e \in U\}$ is all the entities in the universe in the KB. If PMI>0, then T_1 and T_2 are likely correlated and overlapping types. For instance, there are a substantial number of instances belonging to both "educational institution" and "organization", resulting in a positive PMI score. As such, we will not use *any* entity e of type "educational institution" as negative example of "organization", irrespective of whether e has type "organization". Formally, we use $\{e|e \notin T_1, \forall T_2 \ni e, PMI(T_1, T_2) < 0\}$ as the negative examples of T_1.

4.2 Type-detection Models

Figure 4 shows our Hierarchical entity-typing model (referred to as Hi-ET for short), for attribute type detection. We follow a similar hierarchical structure as the Hi-EM model.

We initially try to combine the type-detection task with the attribute-level EM task, using multi-task learning [23], given that they are similar intuitively. However our experiments show inferior quality in both tasks. We believe the reason lies in the fact that the "importance" of tokens in these two tasks are in fact opposite – to detect types for "University of California ", the token "University" is more important, but for EM (e.g., compare to "University of Colorado"), "California" is actually more important. Because of this reason we will design and train different models for type-detection.

[2]This problem of missing types can arise because types in KB are often generated from various sources (e.g., structured feeds, web data, etc.), which often do not cover Less popular entities as well as the head entities.

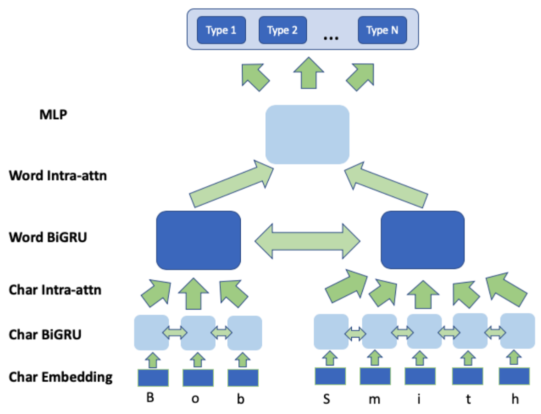


Figure 4: Our Hie-ET model to detect attribute types

Our type-detection model predicts 40 common KB types plus 9 address types. Since it can detect more than one type for each input value, we use one binary classifier for each target KB type, which is connected to the last layer of our model (Figure 4), for a total of 49 such classifiers. A benefit of this setup is that it can easily extend to new types, without needing to re-train existing models that have been tuned and tested.

The model in Figure 4 takes an input attribute value with n words $w_i, i \in [1, n]$, where each word contains l_i characters, written as $c_{it}, t \in [1, l_i]$. It produces C binary labels $(o_1, ...o_C) \in \{0, 1\}^C$ for the C pre-trained types. We describe each component of the model in Figure 4 below, using references to definitions in the attribute-level EM model from Section 3.

Word Level Representation. For each character $c_{it}, t \in [1, l_i]$, we first embed it into vectors e_{it} using Eq. 1 in Section 3. Then we use BiGRU layer (Eq. 2) to get contextual character representation hc_{it}. Using hc_{it}, we apply intra-attention (Eq. 6) to weight each contextual hidden state by importance. The final representation for each word r_i is a weighted average of representation hc_{it} (Eq. 7).

Attribute Level Representation. For each word representation $r_i, i \in [1, n]$, we first use another BiGRU layer (Eq. 8) to get contextual representation hw_i. Then we apply intra-attention (Eq. 12) to weight each contextual hidden state by importance. The final attribute representation z is weighted average of word contextual representation hw_i (Eq. 13).

Prediction layer. The representation z passes through an MLP layer. In our model, each binary output has it own MLP layer.

$$o_i = MLP_i(z), i \in [1, C] \tag{16}$$

Then the final output is the softmax of MLP output.

$$p_i = \text{softmax}(o_i) \tag{17}$$

Where $p_i = [p_{i0}, p_{i1}]$ indicates the probability of predicting the input value as the ith pre-trained type as true, and false, respectively.

During training, we use cross-entropy as our loss function, the final loss is the average of C classes over all examples.

$$L = -\frac{1}{N}\frac{1}{C}\sum_{ex}\sum_{i=1}^{C} y_{i0}log(p_{i0}) + y_{i1}log(p_{i1}) \tag{18}$$

Note that the model predicts types for one input value at a time. When predicting types for a column of k values, we simply compute an average score for the k values.

4.3 Transfer Learning for Type Detection

In our proposed EM system, we apply transfer-learning approach by directly using pre-trained type-detection models to detect if any table column/attribute corresponds to one of the known types.

Similar to transfer-learning in attribute-level EM (Section 3.3), we can also apply type-detection models to new types by first building a *unified-model*, which is trained using the union of data for all known types. For a new attribute type, we start from the representation of the unified-model, and use fine-tuning to produce an accurate model for a new type with little training data. Our experiments suggest that this is indeed the case – transfer-learning converges to high-quality type-detection models substantially faster than training from scratch.

5 EXPERIMENTS

In this section, we report experiments on different system components: Type Detection, Attribute-level EM and Table-level EM.

5.1 Type Detection Experiments

The goal of type detection is to accurately predict attribute types using pre-trained models. We report results on three experiments, entity-value type detection, table-column type detection, and transfer-learning for new types.

5.1.1 *Entity-value type-detection.*

Experimental setup. As discussed in Section 3.1, we use pre-trained models to detect 49 attribute-types, which include 40 common entity types from a KB, and 9 address types of different locales/markets (“en-us”, “en-gb”, etc.) from the “Maps” vertical of a search engine.

For each type T, we sub-sample at most 20K entities in T as positive examples (when a type has less than 20K entities we use all). For negative examples of T, we use entities not in T, filtered by the PMI procedure discussed in Section 4.1. This is to filter away negative examples that may be incorrect due to missing type labels (e.g., entity “Gonzaga University” has the type “educational institution” but is missing the type label “organization” in the KB. With PMI filtering we would not incorrectly use “Gonzaga University” as a negative example for “organization”, since the two types are identified as overlapping/related). We randomly split the data into training (80%), development (10%) and test (10%).

We use PyTorch 0.4.1[55] to implement Hi-ET. We use random character embedding initialization with size 300. We use bidirectional GRU with 2 layers, and the size of each hidden layer is 300. For MLP, we use 2 linear layers of size 300 and ReLU as the non-linear activation function.

For training, we use batch size 32 and set dropout rate to 0.4 to help regularization. We use Adam [43] as the optimizer and use the default initial learning rate 0.001. We set gradient clipping threshold to 5.0 to increase stability. We finish the training after 5 epochs.

Experimental results. Figure 5 shows the precision-recall curves of entity-value type detection for the 49 types. In Figure 5(a) and 5(b), we can see that Hi-ET has high precision and recall for most of KB types, showing its ability to differentiate between entity values of different types. There are a few types (computer, architecture venue, airline and sports facility) where the results are not as good. We found a main reason is the lack of positive training data – these types are small with less than 2000 entities in KB, which makes it difficult for deep models to learn. This is further exacerbated by the fact that entities from small types tend to be less popular and have more missing type information: if $e \in T_1$ but if the T_1 type is missing for e in the KB, we will incorrectly use e as a negative example of T_1, confusing the model. For small types and less popular entities, this tends to be more common, and is more difficult for our PMI-filtering approach to detect (Section 4.1).

Figure 5(c) shows precision/recall on 9 address types. It can be seen that despite the subtle differences of addresses from different markets (en-us, en-gb, en-ca, en-nz, etc.), where we intentionally remove obvious indicators such as all country tokens, our models still successfully differentiate addresses between different markets.

5.1.2 *Table-column type-detection.*

Experimental setup. For the table-column type-detection experiment, we use 1M Wikipedia tables as the test set, and evaluate precision/recall of two alternative methods: (1) our pre-trained Hi-ET models, which predict types using the average score of the first 10 values of each column; and (2) a keyword-based approach, which detects types based on keyword in Wikipedia table column-header (e.g., if a column-header contains the keyword "city" or "town", it is predicted to be of type city). Note that keyword is a strong baseline on Wikipedia, as Wikipedia tables are collaboratively edited by millions of editors [31], where column names are well-curated. In comparison, in enterprise CSV files and database tables, column headers are more likely to be cryptic or outright missing [24], which would make keyword search less effective.

We manually label 100 randomly selected columns, detected to be of type T. We report precision results from the 10 most common types in the interest of space.

Experimental results. Figure 7 shows that for most types (7/10), Hi-ET model has comparable or better results. However, quality results from Hi-ET can also be inferior to Keyword Search, notably for the entity type "food". Our analysis suggest that there is little sub-word pattern for this type (e.g., between apple, orange and banana), which is difficult for Hi-ET to generalize.

While Hi-ET is competitive for type-detection, we believe an ensemble of type-detection techniques that combine model-based, keyword-based, and even program-based [60, 72] approaches would be needed for best detection quality in practice.

5.1.3 *Transfer-learning to new types.*

Experimental setup. We also experiment whether our pre-trained type-detection models can be used in transfer-learning, to learn type-detection for other types faster and with less training examples. For this experiment, we use the same data from entity-value type detection (Section 5.1.1). We then select one type out of 49 types as the target-type for transfer-learning, and the remaining 48 types for pre-training. We compare two methods: (1) transfer-learning, using models fine-tuned from pre-trained type-detection models on 48 other types; and (2) learn-from-scratch, without using pre-trained models. For each method, we provide 200, 500 and 2000 examples from the target-type as training, and compare the resulting precision/recall curves.

Experimental results. Figure 6 compares the results with and without transfer-learning on 3 representative types. Similar results are observed in all other types (omitted here due to space constraints). We can see that with transfer-learning from pre-trained models, the model can learn a lot faster compared to learning-from-scratch, especially with little training data (e.g., 200 examples). We can see that results are better for types address and person, since these types have more regularity and are easier to learn. Data in type organization is more complex with more variations, which makes transfer-learning converge slower than other types.

Table 1: Statistics of types for attribute-level EM.

	Train	Dev	Test
Person	921230	3000	112312
Organization	271376	3000	31930
Movie	219574	3000	28832
Location	613792	3000	74062
Organism	5007	1629	1579
Local	85760	3000	10346
Book	21394	2745	2668
Software	2930	348	314

5.2 Attribute-Level Entity Matching

We conduct two experiments for attribute-level EM, pre-trained EM for known types, and transfer-learning for new types.

5.2.1 *Experimental setup.*

Data sets. As discussed in Section 3.1, for each entity e, we use synonymous names of e in KB (from the "alias" attribute, such as "Bill Gates" and "William Gates") as positive examples, and names of a different e' whose name is similar to e (by syntactic distance) as negative examples (e.g., "Bill Gates" and "Bill Clinton"). We split training pairs into train (80%) development (10%), and test (10%).

To evaluate pre-trained attribute-level EM of known types, we use 4 representative types: person, organization, location and movie, which show different types of name variations. The unified attribute-level model is trained using the union of these data.

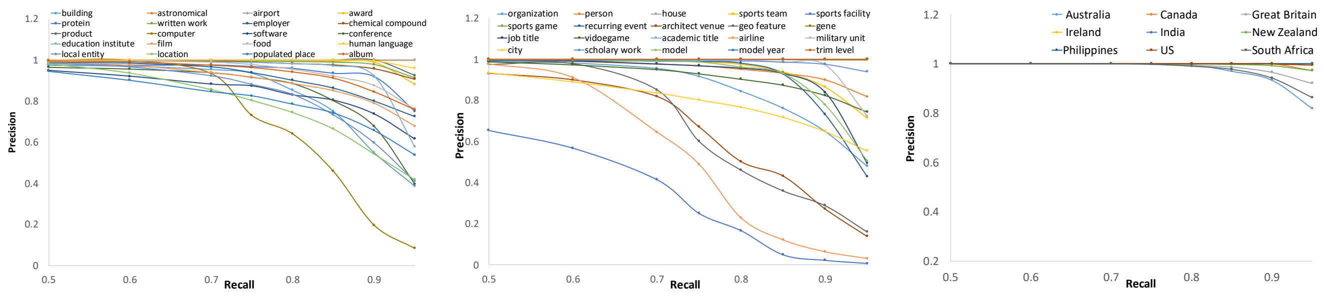
For transfer-learning, we start from the unified model, and report results on 4 different types: organism, local, book and software. We report results after fine-tuning using 200, 500 and 2000 labeled examples. Table 1 reports statistics of these types.

We evaluate the model quality using two metrics: Mean Reciprocal Rank (MRR) and precision/recall.

Methods compared. We compare the folowing methods:

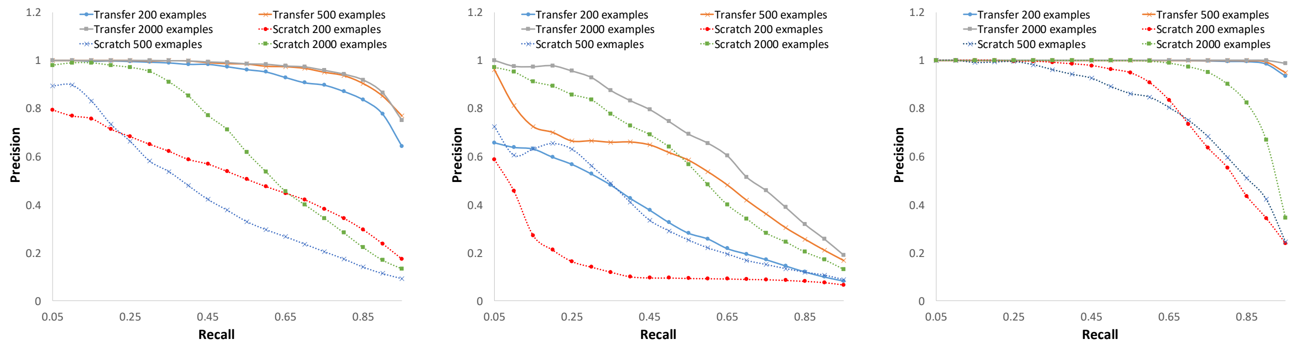
- **DSSM** [38] is one of the first deep models proposed for semantic similarity. DSSM uses DNN to represent each input in a continuous semantic space.
- **DeepER** [40] is proposed to use pre-trained word embedding for EM tasks.
- **DeepMatcher** [49] is also a deep model for EM problem with state-of-the-art results. We use its attribute matching component for attribute-level EM.
- **DeepMatcher (Unified)** is the same as DeepMatcher, but trained on unified data (union of data in different types).
- **Hi-EM** is the proposed EM model with hierarchical deep structure, trained using data for each type.
- **Hi-EM (Unified)** is Hi-EM trained on unified data.

Implementation details. To make fair comparison, we adopt same model settings for all methods. We set 300 as character/word



(a) First 20 Types from KB (b) 20 More Types from KB (c) 9 Types for Address in different locale

Figure 5: P/R curves of entity-value type-detection using Hi-ET model, for 40 KB entity types and 9 address types.



(a) Person (b) Organization (c) US Address

Figure 6: P/R curves for transfer learning using Hi-ET model, varying the amount of training data.

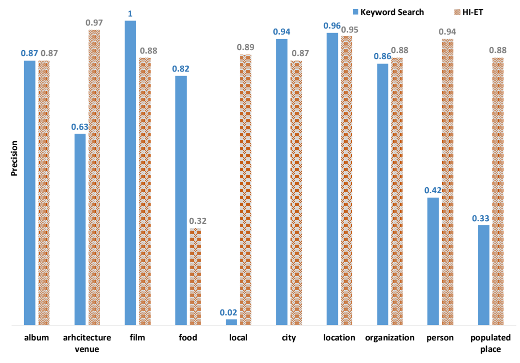


Figure 7: Comparision of Hi-ET and Keyword Search for Table-column type-detection across 10 most common types.

embedding size, 2 layers of size-300 for bidirectional GRU component, 2 layers of size 300 for MLP, and ReLU as nonlinear activation. For training, we use batch size of 32 and set dropout rate of 0.4. We use Adam as the optimizer with the default initial learning rate 0.001, and we set gradient clipping threshold at 5.0. We finish the training after 5 epochs (2 epochs for the unified data set). For each positive entity pair, we randomly select 5 negative examples for negative sampling.

For DSSM, we use character embedding, BiGRU for entity representations, and cosine similarity to compute match scores.

For DeepER, we use Glove [57] as the pre-trained word embedding, and fine-tune the embedding weights during training. For unknown words, we replace with 'UNK' token. We use BiGRU to get entity representations and MLP for final predictions.

For DeepMatcher, the authors present a design space of possible choices (with different neural network component). We report the configuration with the best performance in our experiments, which has character embedding, BiGRU component for attribute summarization, learnable distance (dot product) for attribute comparison and MLP for final predictions.

5.2.2 *Experimental Results.*

Pre-trained attribute-level EM for known types. Table 2 shows the MRR scores for attribute-level EM using pre-trained models on four different attribute types. The proposed Hi-EM produces better results than all other methods. DeepER does not perform well in this task, with lowest scores across all types. The reason is that DeepER uses word-based embedding with many OOV tokens. DSSM also has lower quality than DeepMatcher and Hi-EM, since it uses simple model architecture that does not consider interactions between two input strings. Finally, Hi-EM outperforms DeepMatcher in all types, showing the advantages of the hierarchical architecture, especially on complex attribute-types.

In the same table, we can see that Hi-EM (Unified) also achieves better quality than DeepMatcher (Unified). As expected, the unified models produce lower scores compared to the type-specific models.

Figure 8 reports the same experiments as above using precision/recall curves instead of MRR. The result is consistent with that

Table 2: MRR results for pre-trained attribute-level EM

	Person	Organization	Movie	Location
DSSM	0.888	0.850	0.844	0.853
Deep-ER	0.645	0.528	0.492	0.636
DeepMatcher	0.935	0.909	0.895	0.905
DeepMatcher (Unified)	0.924	0.893	0.894	0.896
Hi-EM	**0.943**	**0.925**	**0.924**	**0.911**
Hi-EM (Unified)	0.934	0.907	0.914	0.899

Table 3: MRR results for transfer-learning to new types

	Organism	Local	Book	Software
200 examples scratch	0.726	0.554	0.612	0.553
500 examples scratch	0.794	0.679	0.701	0.786
2000 examples scratch	0.828	0.804	0.790	0.810
0 example transfer	0.831	0.756	0.863	0.918
200 examples transfer	0.873	0.851	0.865	0.903
500 examples transfer	**0.884**	**0.853**	0.871	0.915
2000 examples transfer	0.881	0.849	**0.880**	**0.937**

of MRR, except in the movie type, where Hi-EM slightly underperforms DeepMatcher in some regions of the curve. An inspection of the errors suggest that the movie type has more synonymous name pairs that are semantic in nature – for example, the movie "Love Song" is also known as "Comrades: Almost a Love Story"; and "Star Crash" is also known as "Star Battle Encounters" or "Stella Star". These positive examples are all very specific and hard to generalize. At training time, Hi-EM overfits on these semantic examples, and produces high match scores for certain negative examples from the test data. Note that because the corresponding true-positive pairs have even higher match scores, in the MRR evaluation these high-scoring negative examples would not affect results. The P/R evaluation on the other hand, are more sensitive to the high-scoring negative examples, which affects precision.

We find other types of errors include name pairs that are inherently ambiguous (e.g., name pairs like "Rick Baker" and "Richard A. Baker" are marked as negative, but similar pairs like "Rick Barnes" and "Richard D. Barnes" would also be marked as positive), which makes it difficult to predict accurately. Finally abbreviation are also difficult to predict – Hi-EM is able to predict some pairs correctly (e.g., "IBM" and "International Business Machines Corp."), but get others wrong (e.g, "University of Geneva" and "UNIGE").

Transfer-learning for new types. Table 3 compares the MRR results between transfer-learning and train-from-scratch, with varying numbers of training data. We note that transfer-learning clearly helps, as there is a significant difference between k-example-transfer and k-example-scratch (for the same k). The difference is more pronounced when using fewer training examples. We also note that for many types (Organism, Book and Software), results from pre-trained unified-model (the line marked as "0 example transfer") already outperforms learn-from-scratch with 2000 examples, which is all the training examples we provide in this experiment.

Figure 9 reports precision/recall curves of the same experiment. We observe that these results are consistent with the MRR results.

5.3 Table-level Entity Matching

In this section, we evaluate our end-to-end EM on table data and compare with existing EM methods.

5.3.1 Experiment Setup.

Methods compared. We compare the following EM methods:

- **Magellan** [44] is a state-of-the-art feature-based EM system. We obtain it from GitHub[3] and use default settings.
- **DeepMatcher** [49] is a state-of-the-art deep EM model. We train the model from scratch, and use the same settings for attribute representations from attribute-level EM.
- **Hi-EM** is our hierarchical EM model trained from scratch, using the same settings from attribute-level EM.
- **Hi-EM (Unified)** is the same as Hi-EM, except that for all attributes we start with representations from the same unified attribute-level EM models that are pre-trained, and fine tune table-level EM based on table-level training data.
- **Hi-EM (Type)** is the same as Hi-EM, except that when attributes are detected as known types, we start with representations from type-specific attribute-level EM models, and fine tune table-level EM based on training data. This is the same as our end-to-end architecture outlined in Figure 2.

For both Hi-EM and DeepMatcher, we concatenate the representations of all attribute pairs and apply a 2 layer MLP of size 300 for final predictions.

Data sets. We use labeled data from a repository of EM tasks[4], which were also used in prior work [44]. There are a total of 24 EM tasks, each with a pair of tables, where some of the record pairs between the two tables were manually labeled as match/non-match. We exclude tasks whose corresponding data have quality issues (mainly due to formatting) and could not be run using the existing Magellan system, and ones that are very easy (e.g. matches are almost all exact, and all methods have over 0.95 $F1$). We use 8 remaining data sets that are more challenging EM tasks.

We evaluate model performance with varying number of training data. Specifically, for each data set, we randomly sample 5%, 10% and 20% labeled data as training, and use the remaining 80% as testing. We report $F1$ score on the test data. Note that the training data in most cases translate into just a few dozen labeled record pairs (there is one with only 13 training pairs), which is a small amount of training data.

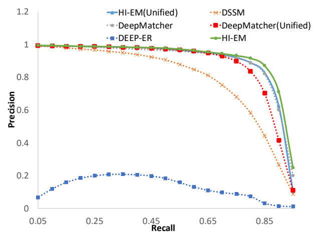
To reduce randomness, we run deep models three times with different random seeds, and report an average $F1$. We keep the same random seed in each run between different deep models.

5.3.2 Experimental Results.

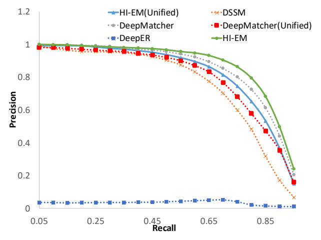
Table 4 shows $F1$ score across different data sets with varying amounts of training data. First, the Hi-EM (Type) achieves better quality than Hi-EM (Scratch) in 23 out of 24 settings, and it outperforms DeepMacher in 22 out of 24 settings, showing the benefit of a pre-training approach even with a small amount of training data.

Between Hi-EM (Type) and Hi-EM (Unified), Hi-EM (Type) produces better quality in 12/24 settings, and there are 6/24 settings for which the two methods are identical (since no columns are detected to be of known types). This is consistent with our finding in attribute-level EM that type-specific models are more accurate.

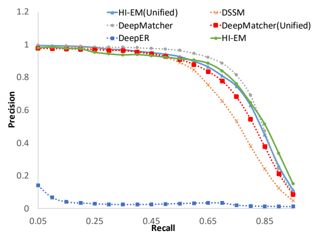
[3]https://github.com/anhaidgroup/py_stringmatching
[4]Available at https://sites.google.com/site/anhaidgroup/useful-stuff/data



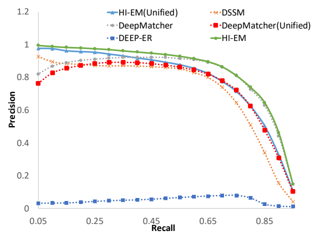
(a) Person



(b) Organization

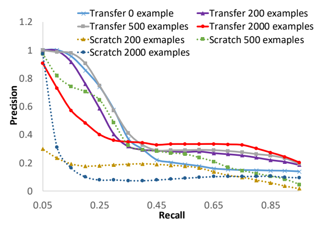


(c) Movie

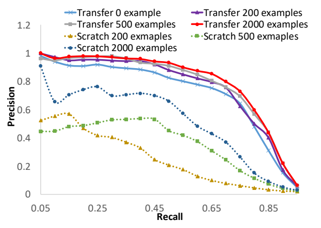


(d) Location

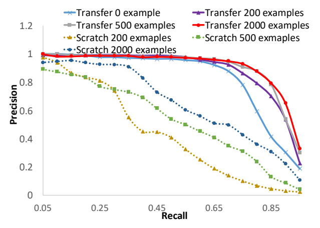
Figure 8: P/R curves of different models for attribute-level EM, on 4 types of attributes.



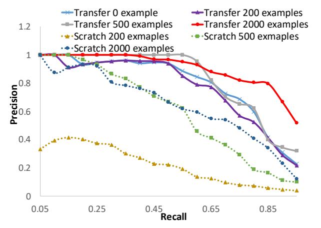
(a) Organism



(b) Local



(c) Book



(d) Software

Figure 9: P/R curves of transfer-learning for new types using Hi-EM, with varying training data for 4 different attribute-types.

Table 4: F1 for table-level EM with varying training data

	Hi-EM Type	Hi-EM Unified	Hi-EM Scratch	Deep-Matcher	Magellan
Amz-BN 5%	0.510	0.450	0.460	0.200	**0.720**
Amz-BN 10%	**0.853**	0.810	0.657	0.350	0.820
Amz-BN 20%	**0.854**	0.790	0.646	0.650	0.800
Bk-Bw 5%	**0.820**	**0.820**	0.517	0.750	0.250
Bk-Bw 10%	**0.890**	**0.890**	0.820	0.850	0.400
Bk-Bw 20%	0.790	0.790	0.854	**0.880**	0.410
GR-BN 5%	**0.778**	0.720	0.612	0.680	0.710
GR-BN 10%	**0.809**	0.780	0.634	0.670	0.680
GR-BN 20%	**0.843**	0.830	0.661	0.650	0.650
YP-Yelp 5%	**0.955**	0.930	0.755	0.850	0.810
YP-Yelp 10%	**0.955**	0.940	0.721	0.850	0.810
YP-Yelp 20%	0.950	**0.970**	0.792	0.860	0.910
Amz-RT 5%	**0.852**	0.825	0.617	0.810	0.822
Amz-RT 10%	**0.868**	0.861	0.679	0.812	0.833
Amz-RT 20%	**0.853**	0.845	0.648	0.773	0.833
ANI-MAL 5%	**0.989**	**0.989**	0.927	0.956	0.633
ANI-MAL 10%	**0.989**	**0.989**	0.967	0.963	0.518
ANI-MAL 20%	**0.989**	**0.989**	0.967	0.964	0.481
BN-Half 5%	**0.932**	0.929	0.917	0.913	0.677
BN-Half 10%	0.936	**0.939**	0.919	0.917	0.744
BN-Half 20%	0.940	**0.953**	0.905	0.908	0.899
RE-IMDB 5%	0.803	0.811	0.772	0.803	**0.899**
RE-IMDB 10%	0.880	**0.885**	0.756	0.883	0.881
RE-IMDB 20%	0.863	0.904	0.842	0.860	**0.937**

Compared to the feature-based EM Magellan, Hi-EM (Type) outperforms Magellan in 20/24 settings. On the other hand, DeepMatcher is comparable to Magellan, which is consistent with what is reported in [49].

6 RELATED WORKS

In this section, we describe existing work in three related areas: Entity Matching, Deep Learning in NLP, and Transfer Learning.

Entity Matching. Entity matching, also known as entity resolution, fuzzy join, record linkage, among other names, has been a long-standing problem in the literature of data mining and data integration [28, 30, 33, 45]. Various techniques have been proposed, including ML-based approaches [10, 15, 61, 65] (based on positive/negative pairs labeled by users), and constraint-based methods [11, 19, 63, 70] (based on rules programmed by experts). Recently, two deep EM models, DeepMatcher [40], and DeepER [49], have been proposed and are shown to achieve better result quality.

We observe that most existing EM approaches require a large amount of training data, which is a significant barrier to wider adoption. We in this work propose a hands-off Auto-EM architecture, which leverages type-detection and attribute-level EM models that are pre-trained on a large amount of data from KBs. It is shown to achieve high EM quality with little training data.

Entity type detection. Unlike the literature on type classification in NLP (e.g., [50, 75]), which typically relies on natural language contexts, our task of entity type detection is for database tables and columns, where natural language contexts are absent. Our approach leverages only characteristics of entities, in a setting similar to [72].

Deep model in NLP. Tremendous progress have been made in applying deep models to NLP. Text-classification and similarity problems are particularly relevant to our problem.

Classification. Text classification [9] is a fundamental problem in NLP, for which neural network models are developed, including Convolution Neural Network (CNN) [42] and Recurrent Neural Network (RNN)[52]. Recently self-attention [68] is used as additional layer to improve performance, and [74] adopts hierarechical structure with self-attention over RNN for document classification.

Similarity. Learning textual similarity between two input is important in NLP, with applications including Natural Language Inference (NLI) [17, 71], Answer Selection (AS) [73], etc. Early deep models [38] treat each input independently. Recently approaches leverages input pairs [20, 35, 54, 62], and solve these tasks with a similar architecture of input embedding layer, context encoding layer, interaction and attention layer, and finally output classification layer [47].

Transfer learning. Transfer learning has a long history [53], where the idea is to transfer knowledge from a problem with abundant training data, to a target-problem that is related but with limited data. Transfer learning has been successfully applied to domains such as computer vision and NLP [14].

In NLP, transfer learning approaches include the well-known word embedding [41, 48]. Recent approaches propose pre-train models with language model objectives [25, 37], with fine-tuning for specific tasks, which has achieved great success [27, 59].

7 CONCLUSION AND FUTURE WORK

In this work we propose an end-to-end system for EM, that leverages models pre-trained using rich KB data. With the help of transfer-learning, we show that table-level EM tasks can be trained with little labeled data.

Our deep models are not currently good at detecting attributes involving numeric values (e.g., currency and measurements). Using programmatic methods to detect and featurize these attributes could complement the deep models for better overall EM results, and are interesting directions for future work.

REFERENCES

[1] [n. d.]. AWS Lake Formation ML Transforms: FindMatches/Deduplication. https://aws.amazon.com/lake-formation/faqs/.

[2] [n. d.]. Azure Machine Learning Data Prep SDK. https://docs.microsoft.com/en-us/python/api/overview/azure/dataprep/intro?view=azure-dataprep-py.

[3] [n. d.]. Bing Entity Search API. https://azure.microsoft.com/en-us/services/cognitive-services/bing-entity-search-api/.

[4] [n. d.]. Excel Fuzzy Lookup Addin. https://www.microsoft.com/en-us/download/details.aspx?id=15011.

[5] [n. d.]. Google Knowledge Graphs. https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html.

[6] [n. d.]. Microsoft Dynamics 365 for Customer Insights. https://dynamics.microsoft.com/en-us/ai/.

[7] [n. d.]. Salesforce Customer 360. https://www.salesforce.com/blog/2018/09/what-is-salesforce-customer-360.html.

[8] [n. d.]. Self-Service Data Preparation, Worldwide, 2016. https://www.gartner.com/doc/3204817/forecast-snapshot-selfservice-data-preparation.

[9] Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. In *Mining text data*. Springer, 163–222.

[10] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 783–794.

[11] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-scale deduplication with constraints using dedupalog. In *IEEE International Conference on Data Engineering*. IEEE, 952–963.

[12] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[14] Yoshua Bengio. 2012. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 17–36.

[15] Mikhail Bilenko and Raymond J Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 39–48.

[16] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. AcM, 1247–1250.

[17] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 632–642.

[18] Kaushik Chakrabarti, Surajit Chaudhuri, Zhimin Chen, Kris Ganjam, Yeye He, and W Redmond. 2016. Data services leveraging Bing's data assets. *IEEE Data Eng. Bull.* 39, 3 (2016), 15–28.

[19] Surajit Chaudhuri, Anish Das Sarma, Venkatesh Ganti, and Raghav Kaushik. 2007. Leveraging aggregate constraints for deduplication. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 437–448.

[20] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2017. Enhanced LSTM for Natural Language Inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 1657–1668. https://doi.org/10.18653/v1/P17-1152

[21] Tao Cheng, Hady W Lauw, and Stelios Paparizos. 2012. Entity synonyms for structured web search. *IEEE transactions on knowledge and data engineering* 24, 10 (2012), 1862–1875.

[22] Peter Christen. 2009. Development and user experiences of an open source data cleaning, deduplication and record linkage system. *ACM SIGKDD Explorations Newsletter* 11, 1 (2009), 39–48.

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

[24] Eli Cortez, Philip A Bernstein, Yeye He, and Lev Novik. 2015. Annotating database schemas to help enterprise search. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1936–1939.

[25] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*. 3079–3087.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[28] AnHai Doan and Alon Y Halevy. 2005. Semantic integration research in the database community: A brief survey. *AI magazine* 26, 1 (2005), 83.

[29] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 601–610.

[30] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering* 19, 1 (2007), 1–16.

[31] Yérali Gandica, J Carvalho, and F Sampaio dos Aidos. 2015. Wikipedia editing dynamics. *Physical Review E* 91, 1 (2015), 012824.

[32] Yuqing Gao, Jisheng Liang, Benjamin Han, Mohamed Yakout, and Ahmed Mohamed. 2018. Building a Large-scale, Accurate and Fresh Knowledge Graph. In *KDD*.

[33] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2018–2019.

[34] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 273–278.

[35] Hua He and Jimmy Lin. 2016. Pairwise Word Interaction Modeling with Deep Neural Networks for Semantic Similarity Measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 937–948. https://doi.org/10.18653/v1/N16-1108

[36] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-data-by-example (TDE): An Extensible Search Engine for Data Transformations. *Proc. VLDB Endow.* 11, 10 (June 2018), 1165–1177. https://doi.org/10.14778/3231751.3231766

[37] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 328–339.

[38] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2333–2338.

[39] Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1377–1392.

[40] Muhammad Ebraheem Saravanan Thirumuruganathan Shafiq Joty and Mourad Ouzzani Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018).

[41] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[42] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*

Processing (EMNLP). Association for Computational Linguistics, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[43] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[44] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1197–1208.

[45] Hanna Köpcke and Erhard Rahm. 2010. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering* 69, 2 (2010), 197–210.

[46] Solomon Kullback. 1997. *Information theory and statistics.* Courier Corporation.

[47] Wei Lan, Wuweiand Xu. 2018. Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering. In *Proceedings of the 27th International Conference on Computational Linguistics.* Association for Computational Linguistics, 3890–3902. http://aclweb.org/anthology/C18-1328

[48] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems.* 3111–3119.

[49] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data.* ACM, 19–34.

[50] David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30, 1 (2007), 3–26.

[51] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2016), 11–33.

[52] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. 2016. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 24, 4 (2016), 694–707.

[53] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[54] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A Decomposable Attention Model for Natural Language Inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing.* 2249–2255.

[55] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[56] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web* 8, 3 (2017), 489–508.

[57] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* 1532–1543.

[58] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL.*

[59] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf* (2018).

[60] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.

[61] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 269–278.

[62] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603* (2016).

[63] Warren Shen, Xin Li, and AnHai Doan. 2005. Constraint-based entity matching. In *AAAI.* 862–867.

[64] Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2015), 443–460.

[65] Parag Singla and Pedro Domingos. 2006. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on.* IEEE, 572–582.

[66] Michael Stonebraker and Ihab F Ilyas. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.* 41, 2 (2018), 3–9.

[67] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web.* ACM, 697–706.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems.* 5998–6008.

[69] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[70] Steven Euijong Whang, Omar Benjelloun, and Hector Garcia-Molina. 2009. Generic entity resolution with negative rules. *The VLDB Journal* 18, 6 (2009), 1261.

[71] Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers).* Association for Computational Linguistics, 1112–1122. http://aclweb.org/anthology/N18-1101

[72] Cong Yan and Yeye He. 2018. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data.* ACM, 35–50.

[73] Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* 2013–2018.

[74] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 1480–1489.

[75] Dani Yogatama, Daniel Gillick, and Nevena Lazic. 2015. Embedding methods for fine grained entity type classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Vol. 2. 291–296.

[76] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.