Entity Resolution with Hierarchical Graph Attention Networks

Dezhong Yao*
Yuhong Gu*
gu18168@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Gao Cong
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

Hai Jin*
Xinqiao Lv*
hjin@hust.edu.cn
xqlv@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

ABSTRACT

Entity Resolution (ER) links entities that refer to the same real-world entity from different sources. Existing work usually takes pairs of entities as input and judges those pairs independently. However, there is often interdependence between different pairs of ER decisions, e.g., the entities from the same data source are usually semantically related to each other. Furthermore, current ER approaches are mainly based on attribute similarity comparison, but ignore interdependence between attributes. To address the limits of existing methods, we propose HierGAT, a new method for ER based on a Hierarchical Graph Attention Transformer Network, which can model and exploit the interdependence between different ER decisions. The benefit of our method comes from: 1) The graph attention network model for joint ER decisions; 2) The graph-attention capability to identify the discriminative words from attributes and find the most discriminative attributes. Furthermore, we propose to learn contextual embeddings to enrich word embeddings for better performance. The experimental results on benchmark datasets show that HierGAT outperforms DeepMatcher by up to 32.5% of F1 score and up to 8.7% of F1 score compared with Ditto.

CCS CONCEPTS

• **Information systems** → **Entity resolution**; *Mediators and data integration.*

KEYWORDS

entity resolution, collective entity linking, hierarchical heterogeneous graph, graph attention networks

ACM Reference Format:
Dezhong Yao, Yuhong Gu, Gao Cong, Hai Jin, and Xinqiao Lv. 2022. Entity Resolution with Hierarchical Graph Attention Networks. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517872

*D. Yao, Y. Gu, H. Jin, and X. Lv are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology in Huazhong University of Science and Technology.

1 INTRODUCTION

Entity resolution (ER) aims to find whether two data entities refer to the same entity. Figure 1 shows two pairs of products from two online shops, and ER aims to determine whether each pair of product entries are the same product entity. ER is a fundamental task used in many applications. Nevertheless, ER remains a nontrivial task because natural language texts often have disambiguation difficulty due to the quality of the contextual information and the topical coherence. The following example illustrates the importance of extracting semantic and contextual information for entities.

Example: *Consider the two pairs in Figure 1. Existing ER methods [6, 19, 29] would give incorrect results. The first pair of entries share many common words, and thus would be considered as a matching. However, they refer to two different softwares. This is because the recurrent neural network (RNN)-based models [6, 29] represent the text description with embeddings without giving the discriminative words such as "big data" and "cluster" higher weights. Therefore, it is difficult for RNN-based models to distinguish between the two entries with similar content, leading to incorrect ER results. For the second pair of entries, which are a match, would be labeled as unmatched since words "adobe spark" are important and the words in the description attribute are not. Therefore it is important to identify the discriminative features and assign different weights.*

A key challenge for successful ER is figuring out how to capture semantic and latent context [12, 47, 54]. Recently, the deep embedding model has been investigated for the ER task and has achieved promising results [6, 29]. As shown in the bottom of Figure 1, recent ER models [6, 24, 29] embed the attribute text into a semantic distribution and thus transform the ER problem to a semantic matching problem. However, these recent ER models still have two weaknesses. 1) They typically assume that all words and attributes are equally important for an entity, which often fails to identify the discriminative words and give the important words higher weight. 2) Entity descriptions often include ambiguous phrases. A word in different domains/categories may be polysemous or have different meanings. For example, the word "Giant" may refer to a grocery store in the food category and a bicycle shop in the sports category.

To address the first weakness, we propose to extend the graph neural network method [41, 52] to characterize the importance of words and capture the semantic relationships of a word and its context. Specifically, we propose a new type of graph, called *Hierarchical Heterogeneous Graph (HHG)*, to represent entities, attributes, and words. It can capture entity-entity relationship, attribute-entity relationship, and word-attribute relationship. As a result, it enables us to assign different weights for the same word under different

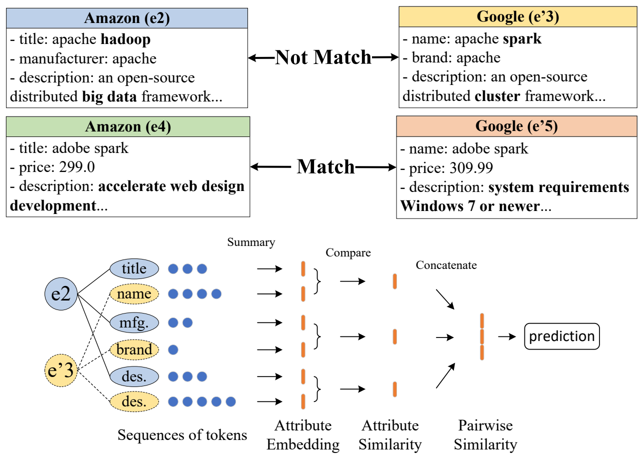

Figure 1: An example of the ER task

categories. To solve the second challenge: polysemous words, we adopt contextual embedding technique. Traditional word embedding techniques learn a global word embedding, which ignores local contextual information. Contextual embedding methods [5, 33] are used to learn sequence-level semantics by considering the sequence of all words in a sentence. Such techniques can learn different representations for polysemous words based on their context.

ER models can be classified into two groups: **pairwise ER models** and **collective ER models** [4]. Pairwise models [6, 24, 29] take pairs of entities as input and resolve ER based on each pair's features independently. As the entities contained in the same document are often semantically related, collective ER models [4, 13, 50] jointly determine entity matching for related entity pairs. However, they resolve ER based on a given knowledge base, which is often not available among heterogeneous data sources.

To overcome the aforementioned weakness, we propose a novel *Hierarchical Graph Attention Transformer* model (HierGAT) based on HHG. HierGAT[1] combines the self-attention mechanism and graph attention network mechanism to solve ER problems for the first time. The self-attention mechanism in Transformer models [5, 39] is effective in capturing the semantic and contextual information of the text, and the hierarchical graph attention model is good at aggregating various types of information to learn better embeddings. First, HierGAT uses the Transformer, which generates highly contextualized embeddings and can identify the discriminative words based on the self-attention mechanism [22]. Second, HierGAT further uses the hierarchical graph attention network model to identify important features for learning better entity embeddings, thus optimizing the matching performance. To the best of our knowledge, this is the first work to successfully exploit the graph attention model to learn embeddings for the ER task. Based on the pairwise ER model HierGAT, we further propose a collective ER model HierGAT+, which jointly determines multiple candidate pairs in one graph whether there is a matching . HierGAT+ is able to exploit the relationship among entities for better accuracy.

The key contributions of this work are summarized as follows:
(1) We define the *Hierarchical Heterogeneous Graph (HHG)* to describe entities as well as their attributes and words, which is able to capture several types of relations, including entity-entity relationship, attribute-entity relationship, and token-attribute relationship. The representation enables us to design a new approach for learning entity embeddings. (Section 2.2)
(2) We propose a novel solution called *Hierarchical Graph Attention Transformer (*HierGAT*)*, which combines the Transformer attention mechanism and hierarchical graph attention network model to learn node representations in HHG. To the best of our knowledge, this is the first work to combine the self-attention and graph-attention mechanisms to solve ER problems. (Sections 4 and 5)
(3) We propose to learn contextual embeddings to enrich the word embedding for better performance. To learn entity similarity embeddings, we introduce the multi-view approach [56] into HierGAT. We translate comparisons between entities into comparisons between their views. (Sections 4 and 5)
(4) We propose a HHG-based collective ER model HierGAT+, in which a query entity and candidates are represented in one graph for ER decision.
(5) We provide an extensive empirical evaluation on both pairwise model HierGAT and collective model HierGAT+ by comparing with several ER models [8, 19, 24, 29], including the state-of-the-art model DeepMatcher and a current work Ditto, on datasets of different sizes and domains, i.e., the Magellan [19] and WDC product matching datasets [35]. Our experimental results demonstrate that HierGAT outperforms DeepMatcher by up to 32.5% of F1 score and Ditto [24] by up to 8.7% of $F1$ score. Moreover, HierGAT maintains robustness on dirty datasets as well as datasets with few labels. The experimental results also show that the collective model HierGAT+ achieves up to 7.1% $F1$ improvement than Ditto and up to 6.4% improvement than pairwise model HierGAT. (Section 6)

2 PROBLEM FORMULATION

We first define the ER problem and then describe the use of HHG structure as well as the construction of HHG.

2.1 Entity Resolution

In the ER problem, an entity often represents a real-world object, such as product, person, company, etc. Each entity e is described by pairs of $<key, val>$, where key and val denote the name and value of an entity attribute, respectively. To handle incomplete data, the missing attributes are filled with word "NAN".

ER Problem: *Given two collections of data entities D and D', the goal of ER problem is to output an entity matching matrix $L \subseteq D \times D'$, where element $l_{ij} = \{(e_i, e_j) | e_i \in D, e_j \in D'\}$ indicates whether e_i and e_j match.*

To solve this problem, a straightforward way is to take a query entity e_q from D and then compare it with all the candidate entities in D'. Most of the existing methods [6, 24, 28, 29] take N entities pairs $\{(e_q, e_k)|_{1 \le k \le N}\}$ as input and treat those pairs independently, which is referred to as the pairwise ER.

In this paper, we also consider collective ER, in which a query entity has N candidate entities and we determine their matching relationships together. As shown in Figure 2, we create a relation network to describe the matching relations between the query entity e_q and N candidates. The three nodes (e_q vs. e_1), (e_q vs. e_2), (e_q vs. e_3), named *comparison nodes*, are represented by vectors describing the semantic relation between two entities. As shown in

[1]Source code of HierGAT https://github.com/CGCL-codes/HierGAT.

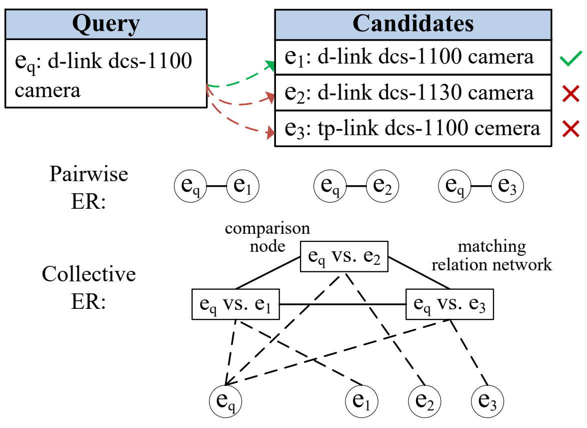
Query
eq: d-link dcs-1100 camera
Candidates
e1: d-link dcs-1100 camera
e2: d-link dcs-1130 camera
e3: tp-link dcs-1100 cemera
Pairwise ER:
Collective ER:
comparison node
matching relation network
eq vs. e1
eq vs. e2
eq vs. e3

Figure 2: Two types of ER systems

the relation network, it is desirable to capture the relation between the matching nodes (e_q vs. e_1) and the other two unmatching nodes during training, making the matching node different from the other two unmatching nodes.

Typically, an ER system consists of two steps: blocking and matching. In this paper, we focus on matching, and the blocking step uses word matching [19] to filter out the unmatching pairs.

2.2 Hierarchical Heterogeneous Graph

To represent the structural information of entities, we design a new *hierarchical heterogeneous graph* (HHG), denoted by $G = (V, R)$, where V and R are the node set and edge (relation) set, respectively. As illustrated in Figure 3, HHG comprises three types of nodes: entity node v^e, attribute node v^a, and token node v^t, and each type forms a layer. Each node in a HHG is represented by a semantic vector of the same length. The vector of a token node is its word embedding. The vector representation of an attribute node is an embedding of its corresponding $<key, val>$ pair, where key is the attribute name, and val is summarized from the text value of the attribute. The vector of an entity node is an embedding that aggregates its attributes' embeddings. The node relation of R comprises three types: (1) Token-Attribute relation, characterizing that attributes are composed of tokens, (2) Attribute-Entity relation, describing that entities are composed of attributes, and 3) Entity-Entity relation, describing the matching relations between entities, i.e., an edge between two entities are established if they match. As the word position in an attribute also has semantic meaning we use the orders of words in the attribute node to represent the word positions, which is inspired by the Transformer techniques.

There are two advantages of the proposed HHG. 1) HHG preserves the hierarchical structure of an entity, and we can learn the entity embedding from bottom to top by following the hierarchy in the HHG. As shown in Figure 3, all the tokens are at the bottom layer. Then each attribute node is linked with multiple tokens. At the top layer, each entity is represented by its attributes. Compared with the pair-wise interdependence model used in the existing ER methods [13], HHG can be considered as a global interdependence model, which captures more contextual information to derive entity embeddings. Based on the HHG, we propose an approach to derive the entity embedding by aggregating through attribute summarization layer and entity summarization layer in Section 5.1. 2) Using

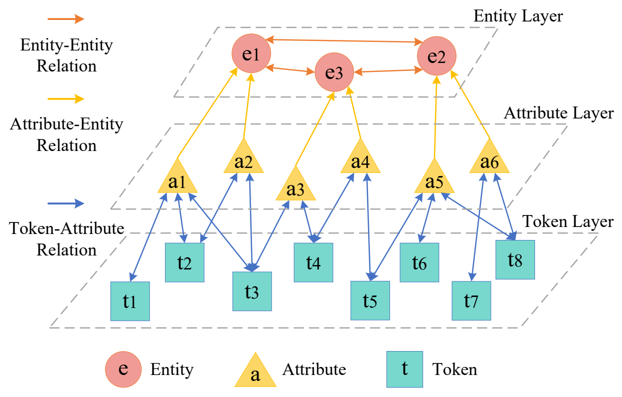

Figure 3: Structure of HHG

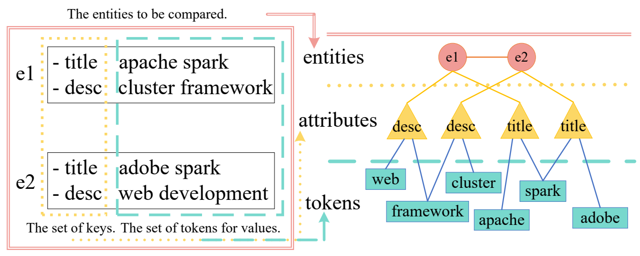

Figure 4: The construction of HHG

HHG, multiple candidate entities corresponding to a query entity can be represented in one graph. The representation is particularly useful for the collective ER decisions, where the query entity e_q and its candidates are connected in the entity layer. Note that the matching context of all nodes is represented in one HHG, which is more comprehensive than separate representations.

HHG Construction: We illustrate the construction of HHG in Figure 4. It takes as input a pair of entities for pairwise ER, or 1+N entities for collective ER. For each input entity, we tokenize its attribute values to get a set of word tokens, and transform the entity into $[< key, [word] >]$ pairs, where each key will become an attribute node v^a and each distinct token will become a token node v^t, and a connection between the attribute node and the token node is established for the pair in the HHG. An entity is formally denoted as $[< v^a, [v^t] >]$, where $v^t \in \mathbb{R}^{1\times F}$ is the word embedding enriched with contextual embedding and F is the dimension size. The attributed embedding $v^a \in \mathbb{R}^{1\times F}$ is calculated by $v^a = \sum h^t(Wv^t)$, where h^t is the self-attention weights of token nodes and $W \in \mathbb{R}^{F\times F}$ is a learnable parameter. The embeddings of entity node v^e are concatenated from its corresponding attributes' embeddings, denoted as $v^e = [v_1^a, ..., v_n^a]$.

Note that each distinct word becomes only one token node in the HHG even if it appears in multiple attributes or multiple entities. For example, there is only one "framework" token node in Figure 4. However, attribute nodes with the same key will not be merged, and thus the keys of attribute nodes are not unique. For example, there are two "desc" attribute nodes in Figure 4, which belong to entities e_1 and e_2, respectively.

3 THE HIERGAT FRAMEWORK

We first present an overview of HierGAT and then present the training process for HierGAT. Figure 5 illustrates the architecture

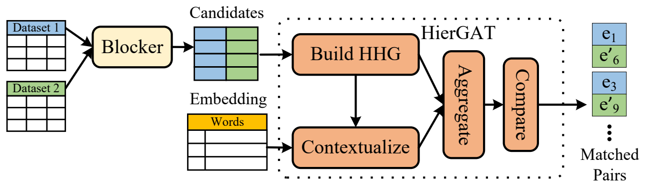

Figure 5: An ER system architecture using HierGAT

of HierGAT and how it fits with a complete ER system, which includes the following three modules:

Blocker: To avoid comparing with all candidate entities, we use existing blocking methods, key-word filtering [19], to prune obviously unmatching pairs. Then we have a reduced set of candidate entities that contain most of the matching entities.

HierGAT: This is the core module, and the focus of this work, which aims to generate the entity embeddings and entity similarity embeddings. As shown in Figure 5, HierGAT has four main components, namely HHG construction, contextual embedding computing, entity aggregation, and entity comparison steps. HHG construction has been covered in Section 2.2. We next focus on the other three components, which are illustrated in Figure 6: (1) *Contextual embedding* (Section 4). This component first generates contextual embeddings by considering three levels of context: token-level context, attribute-level context, and entity-level context. Then the contextual embeddings are used to enrich the original word embeddings to generate word+context (WpC) embeddings. (2) *Entity aggregation* (Section 5.1). This component hierarchically aggregates WpC embeddings to derive entity embeddings based on HHG. The entity aggregation comprises *attribute summarization layer* (Section 5.1.1) and *entity summarization layer* (Section 5.1.2). (3) *Entity comparison* (Section 5.2). This component hierarchically compares the entities' similarities in attribute-level and entity-level to derive entity similarity embeddings. The entity comparison comprises *attribute summarization layer* (Section 5.2.1) and *entity summarization layer* (Section 5.2.2).

Classifier: Similar to previous work, we finally formulate the ER problem as a binary classification problem. We take the results of HierGAT as the input to a classifier to determine whether two entities are a match or non-match.

It is worth noting that the entire pipeline does not require human intervention or additional expert knowledge.

4 COMPUTING CONTEXTUAL EMBEDDING

We propose to learn contextual embeddings to enrich the original word embeddings. The enriched word embeddings will be used as the input for the next phase. We first present the challenges of using word embedding techniques for solving the ER problem, and then our solution of learning contextual embeddings.

4.1 Challenges for Using Word Embedding

Unknown Word: The word embedding models are trained using a large Wiki corpus. However, the embedding of some brand-specific words such as *coolmax* and *tp-link* are still not learned. These words are ignored, but they are very discriminating in the ER problem. Therefore, we should address this.

To address the problem, Glove [32] replaces all unknown words with word "UNK" and learns its embedding for all the unknown words. Obviously, this approach fails to distinguish the unknown

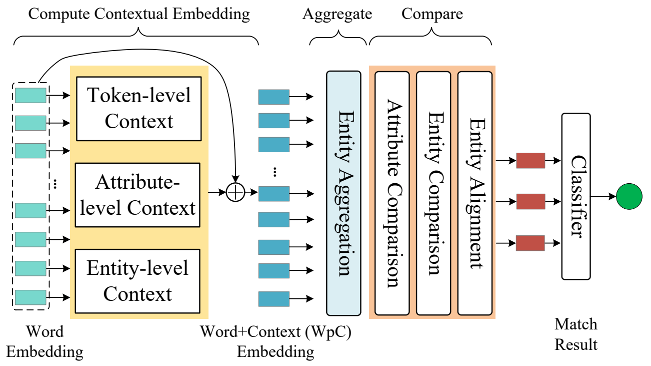

Figure 6: The framework of HierGAT**. Contextual embedding embeds local context to original word embeddings. Entity aggregation generates entity embeddings. Entity comparison calculates the similarity embedding of entities.**

words, which is unsuitable for ER. FastText [2] is a character-level embedding, which constructs an embedding for an unknown word by a sum of the vector representations of its character n-grams. Thus, each unknown word gets a different embedding, but the embedding may not capture the semantic of a word well. DeepER [6] proposes to use the co-occurrence relationship between unknown words and known words to generate embeddings. The embedding of an unknown word is computed by the average of embeddings of the Top-K co-occurrence known words. Nevertheless, the averaging makes common words more influential, and thus embeddings of unknown words tend to be similar.

Word Context Acquisition: The same word token in different attributes or different entities may have different contextual information. For example, the word "Giant" has different semantic meanings in the food category and sport category. However, the recent deep learning based ER solutions learn the same embedding for each word, irrespective of contextual information. To this end, we need to generate different embeddings for the same word under different contexts.

To address the problem, the RNN model [30] provides a way to integrate word embeddings with contextual information from the surrounding words. This allows the same word to have different embeddings when used in different attributes. However, the token-level contextual propagation may not be effective [30] since contextual information can only be propagated sequentially according to the attribute word sequence. The change of token order has a great impact on word embedding. This makes the model susceptible to data heterogeneity. In DeepMatcher [29], an additional alignment operation of shared words for candidate entity pairs is proposed. The operation aims to eliminate the effect of word sequence inconsistencies.

4.2 The Proposed Solution

To address the two aforementioned problems, we argue that it would be important to be able to utilize the contextual information. Our high-level idea includes (1) an unknown word uses its contextual information as the initial embedding value, and a known word can adjust its own embedding according to its contextual information; and (2) the attribute embedding would provide ideal contextual information since attributes contain co-occurrence relations between

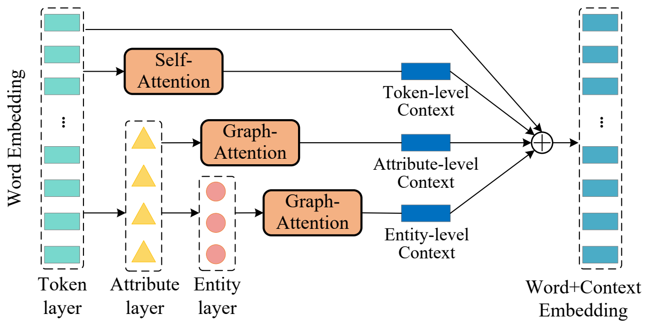

Figure 7: The word+context (WpC) embeddings are calculated by sum up the word embeddings with the three levels of context embeddings.

words and have distinguishable semantic information. Based on the two ideas, we propose two new techniques.

First, we propose to build the HHG (Section 2.2), in which the contextual information propagates across neighbor nodes as it is for *graph neural networks* (GNN). Specifically, attribute nodes can get semantic information from token nodes in the HHG and token nodes can also get contextual information from attribute nodes. Furthermore, a token node can be connected to different attributes of the same entity as well as different attribute nodes of different entities (as shown in Figure 3). Second, we propose to use the attention mechanism to derive distinguishable contextual information for the same word in different contexts. Given a HHG, the token set is denoted as $V^t = (v_1^t, ..., v_{n_t}^t)^\top$, where $V^t \in \mathbb{R}^{n_t \times F}$ and n_t is the number of tokens. The initial word embedding $v_i^t \in V^t$ is from public embeddings, e.g., BERT, and the embedding cannot distinguish the different contexts. In contrast, we use contextual embedding [5] to learn semantics of words and fine-tune the word embeddings with contextual embeddings: $\widehat{V^t}$. Therefore, for the same word we learn different embeddings (and thus semantics) in different contexts.

Furthermore, HierGAT also benefits from the HHG that provides more rich contextual information than the context used in existing work. Unlike the recent ER solutions based on RNN or Transformer models that only consider word context in a sentence, HierGAT exploits three types of contextual information: the token-level context captures semantic meanings in sentences, the attribute-level context captures attributes' semantic, and the entity-level context learns the common words' redundant information. Our contextual embedding approach is illustrated in Figure 7. We proceed to introduce the three types of context embeddings.

Token-Level Context Embedding: The token sequence captures important semantic information, and the meaning of a sentence with different word orders is different. To capture word position information, we use self-attention mechanism to model the relationships between words. The self-attention mechanism can be used to calculate the relevance of the current word to the other words. We use the pre-trained language models such as BERT [5] to extract the token-level context embeddings of all n_t tokens as follow: $C^t = \mathit{Transformer}(V^t) \in \mathbb{R}^{n_t \times F}$.

Attribute-Level Context Embedding: Attribute-Level Context Embedding aims to capture semantic meaning of the attributes of an entity. It identifies the discriminating attributes and gives them higher weights. Meanwhile, it back propagates the weights to the token embeddings. Thus, the same token belonging to different attributes of the same entity or different entities will have different weights. For example in Figure 1, "spark" has higher weight for e_3' than e_4 and e_5'. We propose to use the graph attention mechanism on HHG to model such context. The set of attribute nodes in HHG is denoted as $V^a = (v_1^a, ..., v_{n_a}^a)^\top$, where n_a is the number of attributes in HHG. For an attribute $v_i^a \in V^a$, the set of its adjacent token nodes is denoted as $V_{a_i}^t \subseteq V^t$. We use the vanilla graph attention operation *GraphAttn* [15] to calculate the attribute embedding v_i^a based on $V_{a_i}^t$ by:

$$v_i^a = \mathit{GraphAttn}(c^t, W^t, V_{a_i}^t) \cdot V_{a_i}^t, \tag{1}$$

where $W^t \in \mathbb{R}^{F \times F}$ and $c^t \in \mathbb{R}^F$ are learnable parameters. As shown in Figure 4, multiple attribute nodes may share the same key. We denote the set of unique attributes by $\overline{V^a} = (\overline{v_1^a}, ...\overline{v_j^a}, ...\overline{v_{n_k}^a})^\top$, where n_k is the size of unique attributes. We sum up the attribute embeddings of the attributes with the same key: $\overline{v_j^a} = \sum v_i^a$. The attribute-level context is denoted as a set of embeddings for unique attributes: $C^a = (\overline{v_1^a}, ..., \overline{v_{n_k}^a})^\top \in \mathbb{R}^{n_k \times F}$.

Entity-Level Context Embedding: To handle the collective ER, we represent multiple entities in a HHG, which contains a number of common tokens shared by multiple entities. As one common token appears in multiple attributes, it will be redundantly summed up multiple times when calculating attributes embeddings. The attributes embedding will be biased to the common token, although it may not be an important token. We define these repeatedly summed common token embeddings as redundant context. The impact of the redundant context needs to be reduced. We fist calculate such redundant context C_j^r and apply it as negative contribution to attribute-level context embeddings. The set of common tokens corresponding to unique attribute $\overline{v_j^a}$ is denoted as $\widetilde{V_{a_j}^t}$. First, we compute embeddings of the repeatedly summed common tokens (redundant context) of attribute $\overline{v_j^a}$:

$$C_j^a = \mathit{GraphAttn}(c^a, W^a, \widetilde{V_{a_j}^t}) \cdot \widetilde{V_{a_j}^t}. \tag{2}$$

Next, we want to remove the redundant context from attributed context. As the unique attribute v^a's context is represented as $\overline{V^a}$, the redundant context of attribute v_i^a under the unique attribute set $\overline{V^a}$ is computed as:

$$C_j^r = -\mathit{GraphAttn}(c', (\overline{V^a} || C_j^a)) \cdot \overline{V^a}. \tag{3}$$

The redundant context for all unique attributes is represented as a set: $C^r = (C_1^r, ...C_j^r, ..., C_{nk}^r)^\top \in \mathbb{R}^{n_k \times F}$.

word+context (WpC) embeddings. The contextual embedding is computed as $C = C^t + \Phi(C^a + C^r)$, where Φ is an operation to map attribute embeddings to the corresponding tokens. Then we have the word+context (WpC) embeddings: $\widehat{V^t} = V^t + C$. WpC embeddings will be used to generate entity embeddings.

Training strategy: The training framework of our proposed contextual embedding approach is shown in Figure 6. Different from the previous GNN training process [48, 55], we use bi-directional propagation for training on HHG. This is because the relationship in HHG is bi-directional. *Direction*$_1$:bottom-up information aggregation for attribute embedding; *Direction*$_2$: top-down contextual propagation for word embedding. We use the language model to

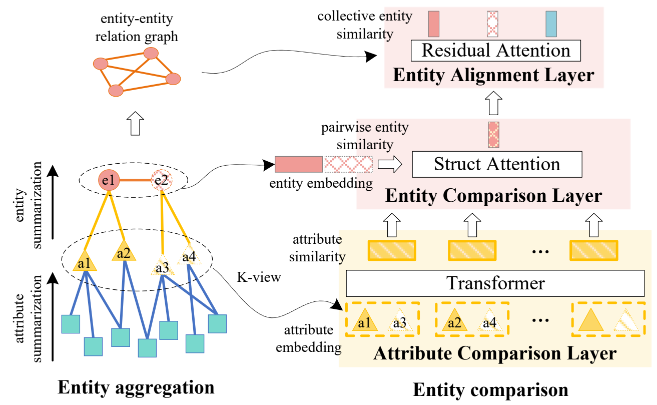

Figure 8: Hierarchical aggregation and comparison. Entity aggregation generates entity embeddings, and entity comparison generates entity similarity embeddings.

get pre-trained word embedding and the pre-trained word embeddings are adjusted during our training process. Therefore, entities will also play the role of contextual information of words, and the information embedded in word embeddings, which can make those tokens more distinguishable under different contexts. For example, the word embeddings of "Giant" in the food category and sport category will be more distinguishable than the embeddings learned without considering entity context. As we use bi-directional propagation on HHG, we introduce the residual mechanism [14] in our model to overcome degradation problem during training.

5 HIERARCHICAL AGGREGATION AND COMPARISON

The context+word (WpC) embeddings $\widehat{V^t}$ and the HHG are taken as input to the hierarchical aggregation and comparison component to compute entity similarity embeddings, which will be used as the input to the classifier to determine the matching. As illustrated in Figure 8, this component includes two steps: (1) hierarchical aggregation to get entity embeddings (Section 5.1), and (2) hierarchical comparison to get entity similarity embeddings (Section 5.2).

5.1 Hierarchical Aggregation for Entity

With the learned word+context (WpC) embeddings as input, we proceed to present how to hierarchically aggregate these embedding based on HHG to get entity embeddings. RNN-based and Transformer-based methods have been used to generate entity embeddings. The RNN-based methods have been shown to advance the accuracy of ER [6, 8, 29]. However, the RNN network fails to incorporate the tokens' importance and position information. Transformer-based approach [24] can overcome these problems, but fails to consider structural information of entities since it combines all attributes into one sentence to learn entity embeddings.

To overcome the shortcomings of the existing solutions, we propose to combine Transformer [39] and *graph attention network* (GAT) mechanism [40] for hierarchical aggregation to obtain entity embeddings. Our method is able to capture the structural information of entities, and distinguish the importance of tokens in learning embeddings. As shown in the left part of Figure 8, the hierarchical aggregation includes two steps to get entity embedding: 1) Attribute Summarization Layer: the attribute embeddings are aggregated from WpC embeddings based on the self-attention mechanism of Transformer. 2) Entity Summarization Layer: the entity embedding is computed by concatenating the embeddings of corresponding attributes.

Algorithm 1: Graph-based Entity Embedding Approach

```
Input  : The HHG $G$ (including entity-attribute $R_{ea}$ &
         attribute-word $R_{at}$ relations) and word+context
         WpC embeddings $\widehat{V^t}$
Output : Entity embedding set $V^e$ of the HHG
1 foreach $i$-th entity $e_i$ in HHG do
2     foreach $k$-th attribute belong entity $e_i$ do
          // Get attribute embedding
3         $V_i^a[k] = Transformer(R_{at}, \widehat{V^t})$
      // Get entity embedding
4     $V^e[i] = Concatenate_k(R_{ea}, V_i^a)$
5 return $V^e$
```

Furthermore, to handle the collective ER, we link the query entity and its candidate entities in a matching relation network as shown in Figure 2. Based on this relation network, we apply the Entity Alignment Layer after the entity summarization layer as shown in Figure 8, which enables us to find the discriminative tokens among the candidate entities collectively. Note that the existing pairwise ER models resolve the pair comparison separately, which does not capture entity interaction.

5.1.1 *Attribute Summarization Layer.*

In this layer, we take as input the WpC embeddings and the Token-Attribute subgraph of HHG, and will generate the attribute embeddings. The Token-Attribute subgraph of HHG is constructed from the attribute keys and values, which contain the tokens in attributes. The challenges for getting attribute embedding are twofold: First, the words in an attribute have different importance. Second, different attributes also have different importance for the same word. Inspired by the self-attention mechanism used in Transformer to distinguish the importance, we use it to distinguish important contextual words for aggregating the embeddings of token nodes. Position encoding is another benefit of using Transformer for aggregation, which captures the sequential information of tokens. The attribute embedding is generated through a sequence of token embeddings. We serialize those token embeddings in the Transformer format as follow:

$$[CLS] token_1\ token_2 \ldots token_n,$$

where [CLS] is the classification token required by the pre-trained *language model* (LM). Then, the Transformer aggregates the token embeddings with attention weights and the final result is in the [CLS] token. We use [CLS] token embedding as the attribute embedding.

5.1.2 *Entity Summarization Layer.*

In this layer, we take the attribute embeddings and the Attribute-Entity subgraph of HHG as input, and generate the entity embedding by concatenating the attribute embeddings. For example, in Figure 8, the embedding of e_1 is obtained by concatenating the embeddings of a_1 and a_2.

The procedure for obtaining the entity embedding is presented in Algorithm 1. Specifically, for k-th attribute, we calculate the attribute embedding by following the solution in Section 5.1.1 (Line 2–3). Then we concatenate all the attributes embedding belonging to the i-th entity e_i to generate entity embedding (Line 4). Finally, We output entity embeddings V^e.

5.2 Hierarchical Comparison for Entity

We have presented how to generate attribute embeddings and entity embeddings in HHG in Section 5.1. We proceed to present how to compute the similarity embedding between entities hierarchically, which will be used as input for the classifier. As illustrated in Figure 8, the comparison has two main steps: attribute comparison and entity comparison.

5.2.1 Attribute Comparison Layer. This layer is to measure the similarity of the attributes of two entities. As shown in Figure 8, given two entities e_1 and e_2, we embed their attributes together as the input to the Transformer in the following format:

$$\{[CLS], e1.v_i^a, [SEP], e2.v_i^a, [SEP]\},$$

where [SEP] is embedding of a special segmentation token and [CLS] is embedding of the classification token required by the pre-trained language model; $e1.v_i^a$ and $e2.v_i^a$ are the attribute embeddings (generated in the Attribute Summarization Layer in Section 5.1.1). The Transformer aggregates the attribute embeddings by self-attention mechanism and we use the embedding of [CLS] token as the attribute similarity embedding. The k-th attribute similarity embedding is denoted as S_k^a.

5.2.2 Entity Comparison Layer. We proceed to present the proposed Structural Attention Operation for the Entity Comparison Layer, which generates the similarity embedding of entities.

A straightforward method of generating entity similarity embeddings is to calculate the similarity based on the concatenations of the entities' attribute similarity embeddings. However, this method does not work well since in the concatenation all attributes are treated with equal importance. For example, in the second pair of products in Figure 1, the title attribute would be more important than the description attribute for comparison.

Inspired by the idea of multi-view learning [44], we consider different attributes as different views of an entity. Each view corresponds to an attribute and contains partial semantic information of the entity. In our model, we consider the following three ways to combine views:

- View Averaging: It treats all the views with equal importance and computes the average of the embeddings of all the views. It does not introduce additional parameters and complex operations.
- Shared Space Learning [56]: It maps all the views to a shared latent view and aggregates them.
- Weight Averaging: It uses the attention mechanism to compute a weight for each view for each entity.

In HierGAT, we use the weight averaging, which is based on the attention mechanisms, as it performs slightly better than the other two options in our preliminary empirical study. We proceed to give more details. For each pair of attributes, we have their similarity embedding S_k^a computed by the attribute comparison layer.

Give two entities to be compared, we concatenate these two entity embeddings $v_{lr}^e = (v_l^e || v_r^e)$ and use concatenated entity embedding v_{lr}^e as the contextual information for attribute similarity calculation. We use the graph attention mechanism to get two entities' similarity representation S_{lr}^e:

$$h_k = softmax(LeakyReLU(c^T(v_{lr}^e || S_k^a)))),$$
$$S_{lr}^e = \sum\nolimits_{k=1}^{K} h_k S_k^a, \tag{4}$$

where $c \in \mathbb{R}^{(1+N)\times F}$ is the weight and h_k is k-th attribute's attention value of two entity's. We compute the weighted sum of the attribute similarity embeddings to obtain the entity similarity embedding S_{lr}^e. Those entity similarity embeddings can be used for a classifier to tell if two entities are a match or not.

5.2.3 Entity Alignment Layer. We design the Entity Alignment Layer for the collective ER model. It aims to compute the similarity embeddings of N entity pairs together based on HHG, where the query entity and N candidate entities are connected in one graph.

Linking multiple entities in one graph enables those entities to learn semantic information from each other. However, multiple entities' attributes may share many common tokens in their descriptions. These tokens may not be important words, e.g., conjunction words and preposition words, but they will increase the similarity scores of two entities. This will have a negative impact on entity matching. Same as redundant context generated by Equation 3. The redundant token embeddings are also aggregated to generate entity embeddings. To alleviate the issue, we propose to use the hard attention mechanism [49] to learn these redundant token embeddings and the remove them from the entity embedding.

Specifically, we remove the redundant token embeddings from each entities by subtract the related entity embeddings which contains the same common tokens. Here, we use the hard attention mechanism to learn weight of the related entity embeddings and the residual calculation [14] to remove the redundant token embeddings. The entity embeddings are updated by attention and residual calculation as follow:

$$h_j = softmax(LeakyReLU(c^T W(v_i^e || v_j^e))),$$
$$\widehat{v_i^e} = v_i^e - W \sum_{j \in D_i} h_j v_j^e, \tag{5}$$

where v_i^e is the i-th entity embedding in HHG; $\{v_j^e, j \in D\}$ represents the entity embeddings that contain redundant token embeddings, and they can be subtracted directly from entity embedding v_i^e to remove the noise; $\widehat{v_i^e}$ is the entity embedding after reducing redundant information and is used for entity comparison.

By using the matching relation network in Figure 2 and entity alignment layer, we extend the pairwise model HierGAT to a collective model named HierGAT+, which can determine the matchings of multiple candidates.

5.3 Training Process

The training procedure of HierGAT is as follows using a labeled training set.

1. Initialize the pre-trained LM and obtain word embedding. Construct the HHGs from the training set;
2. Compute contextual embeddings, and get WpC embeddings;
3. Obtain attribute embedding V^a and entity embedding V^e through hierarchical aggregation;
4. Compute attribute similarity embeddings and entity similarity embedding of two entities;

Table 1: Datasets from Magellan. Datasets annotated with ∗ have a dirty version.

Dataset	Domain	Size	#Pos.	#Attr.
Beer	beer	450	68	4
iTunes-Amazon*(I-A)	music	539	132	8
Fodors-Zagats(F-Z)	restaurant	946	110	6
DBLP-ACM*(D-A)	citation	12,363	2,220	4
DBLP-Scholar*(D-S)	citation	28,707	5,347	4
Amazon-Google(A-G)	software	11,460	11,67	3
Walmart-Amazon*(W-A)	electronics	10,242	962	5
Abt-Buy(A-B)	product	9,575	1,028	3
Company (C)	company	112,632	28,200	1

(5) The final matching results are obtained using the classifier that takes similarity embedding as input and is trained on the training set.

The classifier used for ER is a binary classifier. We use the cross-entropy function to calculate the classification loss. During training our model converges to a state where it fits the dataset well through the back propagation. This process combines the training of the HierGAT with the fine-tuning of the pre-trained LM. Note that, the training process of the contextual embedding is independent of specific pre-trained LM. This makes our proposed ER framework more general and can be adapted to different pre-trained LM.

6 EXPERIMENTS

We report the experimental evaluation for the baseline models and our proposed HierGAT model. These models are implemented in PyTorch and evaluated on a Linux server with a V100 GPU.

6.1 Experimental Settings of Pairwise ER

Benchmark Datasets Following Ditto [24], we use 9 publicly available evaluation sets, which are processed by DeepMatcher [29], and comprise entity pairs after applying blocking. They are originally derived from the Magellan datasets [20] with the number of attributes ranging from 1 to 8. Table 1 summarizes the characteristics of the datasets. The ratios of positive pairs range from 9.4% (Walmart-Amazon) to 25.0% (Company).

We also use four additional dirty datasets that are publicly available from DeepMatcher. In the dirty datasets the entity structure is corrupted by randomly "injecting" attribute values into other attributes. For example, the title attribute may contain the price information. The dirty datasets allow us to evaluate the robustness of the models in real-world scenarios. Finally, the entity pairs in each dataset will be divided into training, validation, and test sets with the ratio of 3:1:1 by following DeepMatcher.

In addition, We use the WDC product matching data, which are extracted from several e-commerce websites and categorized into four different domains: computer, camera, watch, and shoe. Each domain has a test set of size 1100, with 300 positive samples and 900 negative samples. In addition, for each domain WDC provides datasets of different sizes, and each dataset is divided into training and validation sets in the ratio of 4:1. These training sets of different sizes can be used to evaluate performance of the models with respect to the size of training data. Following Ditto [24], the four datasets are combined into the "all" dataset to generate a multi-domain dataset to evaluate the generality of the models. Table 2 summarizes the sizes of these datasets. In WDC, positive samples are generated from the raw data by matching product ID. Negative samples are selected with high text similarity, which increases the difficulty of ER. In the WDC datasets, only the title attribute is aligned, and thus we only use this attribute for matching by following Ditto.

Table 2: Datasets from WDC

Dataset	Small	Medium	Large	xLarge
Computer	2,834	8,094	33,359	68,461
Camera	1,886	5,255	20,036	42,277
Watch	2,255	6,413	27,027	61,569
Shoe	2,063	5,805	22,989	42,429
All	9,038	25,567	103,411	214,746

Baseline Models We compare the proposed HierGAT **(HG)** with the following models.

- **Magellan**: Magellan [19] is a widely used entity resolution tool based on traditional machine learning methods. We use it to train five classifiers (decision tree, random forest, SVM, linear regression, and logistic regression) and then use the validation set to choose the best classifier.
- **DeepMatcher (DM)**: DeepMatcher [29] is a state-of-the-art model based on RNN. It uses the GRU-RNN model to learn the attribute embeddings of entities, which are aggregated for matching.
- **Ditto**: We also compare with Ditto [24], which is a concurrent work, and is based on Transformer. Ditto also comes with some optimizations. However, these optimizations are based on domain knowledge and may not generalize to other datasets. Moreover, they do not always outperform the basic version of Ditto. To be fair, we only compare with the basic version of Ditto as HG does not employ domain knowledge.

Note that in the pairwise ER problem, HierGAT does not use the entity-level context embedding and entity alignment layer. To obtain embedding from text data, different models are used in the previous methods. Magellan generates features for entity pairs using a set of distance functions. DeepMatcher uses FastText as the word embedding model to generate a 300-dimensional word embedding for each word, which follows the original setting of DeepMatcher. For Ditto and HierGAT, we use the Transformer library [46] and support 5 pre-trained language models: BERT [5], RoBERTa, RoBERTa-Large [25], XLNet [51], and DistilBERT [36]. By following the setting of Ditto [24], each word is finally represented as a 768-dimensional word embedding, in addition to 1024 dimensions in the RoBERTa-Large model. The model with the highest F1 score in the validation set is used on the test data for both Ditto and HierGAT. For the F1 scores of DeepMatcher and Ditto, we report the numbers from the original paper, and we also reproduce the results based on the open source of the baseline models.

Experimental Settings In the training phase, we use the learning rate of 1e-5 and run 10 epochs. Each epoch is verified by the validation set to avoid over-fitting and Adam [17] is used in the parameter optimizer. For the Transformer-based model, the maximum length of text sequence is limited to 512. The batch size defaults to 16, except that it is set at 4 for the iTunes-Amazon dataset.

Evaluation Metrics Following previous work [8, 19, 29, 48], we use F1 score as the common evaluation metric.

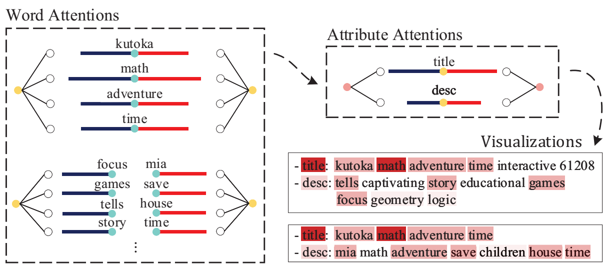
Word Attentions
kutoka
math
adventure
time
focus
mia
games
save
tells
house
story
time
Attribute Attentions
title
desc
Visualizations
- title: kutoka math adventure time interactive 61208
- desc: tells captivating story educational games focus geometry logic
- title: kutoka math adventure time
- desc: mia math adventure save children house time

Figure 9: Attention visualization results of HierGAT **for example entity pairs from Amazon-Google. The darker color indicates the higher attention of** HierGAT.

6.2 Pairwise ER Results

The results on Magellan datasets are presented in Table 4. For both HierGAT and Ditto, we use the evaluation set to select the best of the language models for testing. The last column ΔF_1 indicates the F1 score improvement of HierGAT over the best baseline. As we can see, HierGAT achieves the best result on all datasets. The F1 score improvement is up to 32.5% for DeepMatcher and 8.7% for Ditto. Figure 9 illustrates the effect of attention mechanisms on matching. As we can see, HierGAT gives more importance to discriminating words and important attributes. The attribute "title" and the word "math" are more important for matching judgment.

We also observe that HierGAT performs robustly on the four dirty datasets. On dirty datasets, HierGAT ignores the structure of entities, but its F1 score decreases by only 1% on average. HierGAT still outperforms DeepMatcher and Ditto, and outperforms the best baseline by up to 3.7% in terms of F1 score. The reason could be that the multi-granularity of contextual information overcomes the data heterogeneity problem very well.

Comparing the RNN based model DeepMatcher with the other Transformer-based models, we can see that Transformer offers better performance. To evaluate the generality of our proposed method over different Transformer-based pre-trained language models, we compare with Ditto with respect to language models in three different model sizes (DistilBERT, RoBERTa, and RoBERTa-Large), and the results are summarized in Table 3. HierGAT outperforms Ditto for all the language models. In particular, for the RoBERTa model, the improvement is up to 18.1%. This demonstrates the robustness of HierGAT over the choice of language model. In contrast, Ditto is more affected by the choice of language model, e.g., as the complexity of the model increases, the final result does not get better. We also evaluate the generality of HierGAT for language models on the Dirty dataset. The results further demonstrate the robustness of HierGAT over language models.

Figure 10 shows the results on the WDC dataset with respect to the training data of various sizes. HierGAT and Ditto both use RoBERTa as the language model. HierGAT achieves better results on the WDC dataset consistently over different training data sizes. Notably, the improvement of HierGAT over the baselines is more significant when only a smaller number of training samples is available. On small training set, 1/24 size, HierGAT outperforms Ditto by 6.7% on average. Moreover, on the "all" dataset, with only 1/24 of the training samples (small), HierGAT is able to achieve similar performance as DeepMatcher using all the training samples. This demonstrates that HierGAT is much more label efficient than DeepMatcher and Ditto. HierGAT and Ditto are worse than DeepMatcher on large and xlarge shoe datasets. This could be because shoe datasets have a relatively low proportion of positive samples, for which DeepMatcher uses a positive weight parameter to optimize the results.

Table 3: Differences in F1 scores for three language models

	DBERT			RoBERTa			LRoBERTa		
	Ditto		HG	Ditto		HG	Ditto		HG
Beer	82.5	+5.5	88.0	74.2	+18.1	92.3	90.3	+3.0	93.3
I-A	91.5	+1.1	92.6	92.1	+4.1	96.2	94.3	+2.0	96.3
F-Z	97.3	+2.7	100	98.1	+1.9	100	100	+0.0	100
D-A	98.5	+0.4	98.9	98.9	+0.2	99.1	98.2	+1.0	99.2
D-S	94.9	+0.3	95.2	95.5	+0.5	96.0	95.5	+0.7	96.2
A-G	71.4	+3.3	74.6	65.9	+10.1	76.0	74.3	+2.3	76.8
W-A	79.8	+2.7	82.5	85.8	+2.4	88.2	84.9	+3.6	88.5
A-B	82.5	+1.9	84.4	88.9	+0.9	89.8	92.2	+1.1	93.3
C	48.0	+2.4	50.4	77.8	+4.5	82.3	91.2	+1.7	92.9
Dirty									
I-A	90.1	+2.0	92.1	92.9	+1.7	94.6	87.2	+7.4	94.6
D-A	98.6	+0.2	98.8	98.8	+0.3	99.1	98.7	+0.4	99.1
D-S	94.8	+0.4	95.2	95.4	-0.2	95.2	95.5	+0.2	95.7
W-A	77.9	+0.8	78.7	82.6	+3.7	86.3	85.5	+2.1	87.6

Table 4: F1 scores on the Magellan datasets

Dataset	Magellan	DM	Ditto	HG	ΔF_1
Beer	78.8	72.7	84.6	93.3	+8.7
iTunes-Amazon	91.2	88.5	92.3	96.3	+4.0
Fodors-Zagats	100	100	98.1	100	+0.0
DBLP-ACM	98.4	98.4	99.0	99.1	+0.1
DBLP-Scholar	92.3	94.7	95.8	96.3	+0.5
Amazon-Google	49.1	69.3	74.1	76.4	+2.3
Walmart-Amazon	71.9	67.6	85.8	88.2	+2.4
Abt-Buy	43.6	62.8	88.9	89.8	+0.9
Company	79.8	92.7	87.5	88.2	-4.5
Dirty					
iTunes-Amazon	46.8	79.4	92.9	94.7	+1.8
DBLP-ACM	91.9	98.1	98.9	99.1	+0.2
DBLP-Scholar	82.5	93.8	95.4	95.8	+0.4
Walmart-Amazon	37.4	53.8	82.6	86.3	+3.7

6.3 Collective ER Experimental Settings

Benchmark Datasets The datasets for pairwise ER cannot be used to evaluate collective ER models. We apply a new blocking method on the classical Magellan dataset and the latest DI2KG [34] dataset to generate benchmark datasets. All the datasets in Table 4 have fixed the training and testing sets, that cannot be used for collective ER model. The matching relations for a query entity are randomly split into training and testing sets for pairwise ER. Additionally, only five datasets have public raw data, as summarized in Table 5. Each dataset has two data sources named table A and table B. To

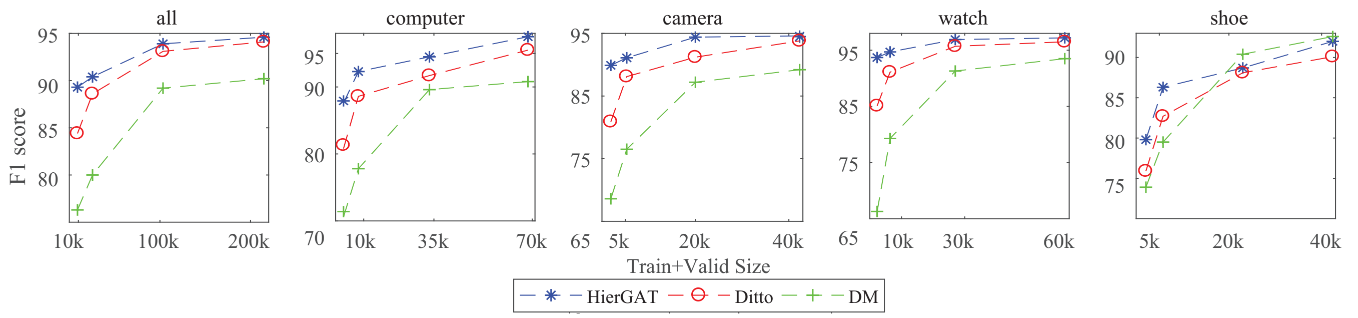

Figure 10: F1 scores on the WDC datasets

Table 5: Sizes of the Magellan datasets

Dataset	Table A	Table B	Candidate
iTunes-Amazon	6,907	55,959	2,295
DBLP-ACM	2,616	2,294	37,740
Amazon-Google	1,363	3,226	19,737
Walmart-Amazon	2,554	22,074	16,354
Abt-Buy	1,081	1,092	17,476

Table 6: Sizes of the DI2KG datasets

Dataset	Table	Product	Candidate
camera	24	29,788	136,260
monitor	26	16,663	310,216

generate training dataset, we randomly select one entity from table A and query top-N similar candidates in table B. We use the TF-IDF cosine similarity to obtain the entities' similarity scores. In this paper, we set N as 16, which can filter out 40% negative samples. The last column in Table 5 summarizes the total number of candidates for all the query entities.

For the DI2KG dataset, the samples are collected from multiple e-commerce websites and categorized into: camera and monitor. As shown in Table 6, there are 24 tables for camera and 26 tables for monitor. To generate candidate set, we select one query entity from dataset and compare it with all the other entities of the same category. Here we also use the TF-IDF cosine similarity to obtain the top-16 similar candidates. Following this blocking method, it generates 136,260 candidates for 29,788 camera entities and 310,216 candidates for 16,663 monitor entities.

For dataset splitting, we use the same split ratio setting 3:1:1 as in Section 6.1. Our data splitting method here is different from the previous method on Magellan dataset in Table 1. Previous split method runs blocking step first and then splits the candidate pairs. For example, given a query entity e_q, the blocking method filters out N candidate entities. Previous split method randomly splits those candidate entities into training and testing sets. This raises a problem that each query entity exists in both training and testing sets. However, in real world applications, we need to handle new unseen entities. Therefore, for the collective ER models, our data splitting method splits the dataset first and runs the blocking step in training and testing sets separately. This ensures that the query entities in the test set do not appear in training set.

Baseline Models In addition to the baseline models used in the evaluation for the pairwise ER models, we also consider five models optimized for the collective ER models.

Table 7: Results of HierGAT+ and baseline methods

Dataset	MG	DM+	GCN	GAT	HGAT	Ditto	HG	HG+	ΔF_1
I-A	50.0	55.9	36.1	36.7	64.6	58.6	59.3	64.7	+0.1
D-A	94.7	98.4	97.4	97.5	98.2	98.8	98.9	99.6	+0.7
A-G	28.5	69.0	64.5	63.6	75.5	77.6	78.0	83.1	+5.1
W-A	58.0	72.5	67.7	54.8	76.7	85.2	85.9	92.3	+6.4
A-B	52.2	62.1	57.6	55.7	68.9	89.3	89.5	93.2	+3.7
camera	-	98.0	82.1	88.2	89.5	99.0	99.1	99.4	+0.3
monitor	-	99.1	78.8	84.0	84.6	98.8	99.2	99.6	+0.4

- **DM+**: We use HierMatcher [8] to optimize DeepMatcher for the collective ER model. The inclusion of hierarchy makes it superior to DeepMatcher on some datasets.
- **GCN**: GCN is the graph embedding method based on spectral domain, which uses the Laplace operator to bring convolution operation into a graph structure.
- **GAT**: Graph Attention Network (GAT) [40] is the graph embedding method, which uses attention mechanisms to control information propagation.
- **HGAT**: We implement the hierarchical information propagation of GAT on HHG. HGAT contains two layers of GAT, the first layer gets the attribute embedding and the second layer gets the entity embedding.
- **HierGAT+ (HG+)**: Based on HierGAT, HierGAT+ further employs the entity-level context and entity alignment operation.

Experimental Settings For the HierGAT+ and HGAT models, we fix 10 epochs of training. The batch size is equal to the candidate set size. So only one query entity will be learned per batch. For the entity-level context, we set the number of common words contained in the context to 10. Moreover, we also use different language models on HierGAT+ to evaluate its generality.

6.4 Collective ER Results

The evaluation results on five Magellan datasets and DI2KG datasets are presented in Table 7. We use the evaluation set to select the best language model for testing. The improvement of HierGAT+ over the second best method on each dataset is given in the last column ΔF_1. We notice that HierGAT+ achieves the best performance on all datasets. HierGAT also outperforms all other baseline methods except for HGAT on iTunes-Amazon dataset.

Comparing Table 4 and Table 7, we notice that the performance of MG, DM, and Ditto drops a lot, especially on iTunes-Amazon dataset. This is because we use a new data splitting method, where the entity in testing set does not exist in training set. The dataset generated by the new split method is more challenging. We find that the performance of HierGAT is only slightly lower than the result on the previous data shown in Table 4.

Table 8: Differences of models in F1 scores for different language models on the Magellan datasets

	DBERT			RoBERTa			LRoBERTa		
	Ditto	HG	HG+	Ditto	HG	HG+	Ditto	HG	HG+
I-A	47.5	57.1	58.2	7.1	11.1	54.2	58.8	61.8	65.6
		+9.6	+1.1		+4.0	+43.1		+3.0	+3.8
D-A	98.8	98.9	99.2	98.2	98.8	99.4	98.9	99.1	99.6
		+0.1	+0.3		+0.6	+0.6		+0.2	+0.5
A-G	75.6	76.4	81.5	77.6	78.0	83.0	78.3	80.7	86.9
		+0.8	+5.1		+0.4	+5.0		+2.4	+6.2
W-A	80.8	81.0	88.6	85.2	85.6	92.3	85.9	90.6	93.9
		+0.2	+7.6		+0.4	+6.7		+4.7	+3.3
A-B	82.6	83.5	92.2	88.3	89.5	92.9	90.9	91.1	94.8
		+0.9	+8.7		+1.2	+3.4		+0.2	+3.7

Table 9: $F1$-score with vs without contextual information

Models	I-A	D-A	A-G	W-A	A-B	camera	monitor
Context	64.7	99.6	83.1	89.2	92.9	99.6	99.4
Non-Entity	63.3	99.4	82.1	88.9	91.9	99.5	99.3
Non-Attribute	64.6	99.4	81.9	88.8	92.2	99.6	99.3
Non-Context	62.6	99.0	81.4	87.8	91.3	99.4	99.0

The effectiveness of hierarchical modeling can be observed by comparing GAT and HGAT. Compared with GAT, HGAT improves by up to 21.9% on Walmart-Amazon dataset. The hierarchical graph structure helps to generate the semantic representation of entity. It uses different attention mechanisms to aggregate the information, so that more semantic information is gathered than the RNN structure. HGAT performs better than MG, DM, GCN, and GAT. This shows the advantages of hierarchical graph modeling with multi-granularity context and attention mechanism.

HGAT does not capture the word order information in the text. With the help of the Transformer, Ditto learns word position information and achieves better performance than HGAT. We note that in iTunes-Amazon dataset, HGAT outperforms Ditto and HierGAT. This could be because the entities in this dataset are music albums, and the attributes are short text. For the attributes with long text, like Abt-Buy, Ditto and HierGAT achieve better results because the word position information is encoded by the Transformer.

HierGAT+ achieves the best results on all datasets since HierGAT+ successfully combines the advantages of graph structure and Transformer. HierGAT+ improve the F1 score by 4.3% on average than HierGAT. The improvement is mainly because HierGAT+ removes the redundant information in the candidate set. By removing the redundant information in the entity-entity relation graph, the entity semantics will be more discriminative. The results on the DI2KG dataset has the same performance on Magellan dataset. Since the Magellan method only supports matching on two datasets, it does not work on the DI2KG dataset. Although the DI2KG dataset is more challenging, HierGAT still outperforms Ditto [24]. HierGAT+ achieves the best performance among all the datasets. From Table 7, we notice that the Transformer-based models Ditto, HG and HG+ perform better than RNN based models. This is because the Transformer learns contextual information and word position information using language models.

To evaluate the generality of our proposed methods for the collective ER task, we compare Ditto, HierGAT, and HierGAT+ with respect to language models in three different model sizes. The evaluation results are summarized in Table 8. It shows that HierGAT outperforms Ditto for all language models. It also shows that HierGAT+ further outperforms HierGAT for all language models. Especially for the RoBERTa model, the improvement is up to 43.1%. This is because HierGAT+ removes the redundant information in entity-entity subgraph. In addition, HierGAT+ improves the independence of the language model. This also demonstrates the robustness of HierGAT+ in solving the collective ER task.

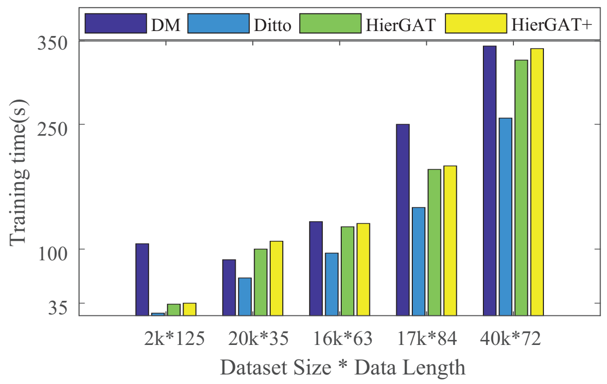

Figure 11: Training time for the Megallan datasets. X-axis is the product of the dataset size and the average data length.

Finally, we evaluate the efficiency of models. The average length of data varies from dataset to dataset. Instead of the size of the dataset, we consider the effect of the average length of the data on the time efficiency. Figure 11 shows that the training time of HierGAT increases linearly. Since DeepMatcher is based on the RNN structure, its running time for long text will be significantly increased. Ditto is most efficient because it does not consider the structure of the data. It serializes all attributes into a single sentence. The structure is preserved in both HierGAT and DeepMatcher, which calculates separately for each attribute. Compared to HierGAT, HierGAT+ takes 3.5% more training time mainly due to entity alignment and removing redundant information.

6.5 Result Analysis

6.5.1 Effect of Contextual Embedding. We evaluate the effectiveness of the proposed methods of modeling context in Section 4.

We compare the following variants: (1) WpC embedding, the proposed embedding methods used in HierGAT+; (2) Non-Entity, WpC embedding without entity context; (3) Non-Attribute, WpC without attribute context; (4)Non-context, WpC without any context, i.e., word embedding only. The results are presented in Table 9. The results show that all the three types of context contribute to the result of HierGAT+. The context allows polysemous and unknown words to be distinguished under different attributes. Moreover, the semantic information in the word embedding fits the context, making the final embedding more discriminating for ER. Additionally, the results also demonstrate the effectiveness of entity-level context to remove redundant information.

6.5.2 Effect of Attribute Summarization. This experiment is to evaluate the effectiveness of the proposed weight averaging method in multi-view learning to learn the entity similarity by comparing our method and the two other combining methods in Section 5.2. The F_1 scores of the three methods are shown in Table 10. The shared

Table 10: $F1$ **score of different attribute summarizations**

Methods	I-A	D-A	A-G	W-A	A-B
View Average	56.1	99.1	75.1	82.3	85.4
Shared Space Learn	55.6	99.0	74.4	81.0	81.8
Weight Average	64.7	99.6	83.1	89.2	92.9

space learning has the worst accuracy among the three methods, whereas weight average based on structured attention achieves the best accuracy. The view averaging achieves better accuracy than the shared space learning method. Because the attributes themselves are composed of tokens, there is no difference in latent semantics and different attributes are embedded in a shared space from the beginning. The shared space learning is used as the multi-layer perception and instead makes the attribute embedding enter a different space, making the result worse. The reason why the weight average based on the structural attention mechanism achieves the best results is that the heterogeneity of data is challenging to find out discriminating attributes for ER. The weight average based on structural attention mechanism enables us to highlight more discriminating attributes, making it easier for the classifier to determine entity matching.

6.5.3 Effect of Hierarchical Comparison Module. We evaluate the effectiveness of the entity comparison components. We compare the following variants: (1) HierGAT+, the complete model; (2) Non-Sum, HierGAT+ without entity summarization component; (3) Non-Align, HierGAT+ without entity alignment component. The results are presented in Table 11. The results show the effectiveness of these components in HierGAT+.

6.6 Summary

The pairwise model HierGAT achieves the best performance on all datasets and improves the F1 score by up to 8.7% over the best baseline. HierGAT achieves better robustness on dirty datasets. We also verify that HierGAT is generalizable on different pre-trained language models. HierGAT achieves stable results on WDC datasets with different amounts of labeled data. HierGAT requires 1/2 of the training samples to obtain similar results as Ditto. Comparing with DeepMatcher, HierGAT only requires 1/8 of the training samples to achieve similar performance.

For the collective model HierGAT+, seven new datasets are constructed to evaluate the effectiveness of the proposed models. The experimental results show that it is effective to maintain the relationships within the candidate set. HierGAT+ improves the F1 score of HierGAT by an average of 4.3% on the Magellan datasets.

7 RELATED WORK

Entity resolution has been extensively studied for decades, and is still receiving continuous attention [6, 11, 38, 53]. Many methods have been proposed to advance the area, which can be categorized into three types: 1) The rule-based methods [7, 37, 43] require designing rules and setting thresholds; 2) The machine learning-based models [1, 19] require features engineering for learning classifiers. 3) Crowdsourcing-based approaches [10, 42] are dependent on substantial human efforts.

Recently, deep learning techniques have been explored to solve the ER problem [21, 27]. A typical approach is to abstract the ER problem as a binary classification problem, such as DeepER [6], DeepMatcher [29], and Ditto [24]. DeepER uses Glove to get word embedding and then uses LSTM [16] to get entity embedding. DeepER also modifies the pre-trained word embedding using back propagation. Inspired by this method, we add the context calculation process to the training phase in our method, where the word embedding and contextual information can be modified by back propagation. DeepMatcher generalizes DeepER to introduce a framework for solving the ER problem. Other models like MPM [9], Seq2Seq [30], and HierMatcher [8] follow this framework with some modifications, which may get better results on some datasets. Transformer structures have been shown to be effective for transfer learning on ER problems [3, 31]. Ditto uses fine-tuned pre-trained Transformer-based language models. Ditto is also equipped with some optimizations, which may need domain knowledge. These deep learning based methods are based on text sequences for matching judgments. They use different serialization methods for attribute embedding and attribute similarity representation.

Table 11: $F1$ **score of aggregation and comparison modules**

Methods	I-A	D-A	A-G	W-A	A-B	camera	monitor
HG+	64.7	99.6	83.1	89.2	92.9	99.6	99.4
Non-Sum	63.5	99.2	82.6	87.9	90.6	99.1	99.2
Non-Align	62.5	99.1	77.1	85.8	86.3	99.3	99.1

Our work is related to graph neural networks, such as GCN [18, 23] and GAT [40, 45], which employ deep learning techniques for graph representation learning. The graph neural networks such as GCN and GAT are designed for graphs in which nodes are rich in features and semantics, which allows node embedding to converge without restricting the propagation of information. In contrast, in the ER problem, only words have initial semantic information and nodes do not have rich semantics. Therefore, it is not suitable to directly apply GCN or GAT to the HHG. To address the challenge, the propagation of information needs to be strictly controlled for different types of nodes. In our method, by placing nodes in different layers of the HHG, we exploit the propagation of information between layers to make entity embedding more discriminating.

Graph neural networks have been applied to the *entity alignment* (EA) problem [26, 48, 55]. The EA problem is related to, but different from the ER problem. In the EA problem, the entity is described by the RDF format and rigorous pre-processing of the original data is required. This makes the models for EA either difficult to apply or inapplicable to the ER problem.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose to use HHG to represent entities, which can capture several types of relationships among entities. We propose a pairwise ER model HierGAT and a collective ER model HierGAT+, which combine the graph attention model with the Transformer. The experimental results show that HierGAT outperforms baselines significantly. Moreover, HierGAT also performs better on dirty datasets or datasets with fewer labels than the existing methods. An interesting future direction is to extend HierGAT to the setting of unaligned attributes.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China under Grant No.62072204, the Fundamental Research Funds for the Central Universities under Grant 2020kfyXJJS019, and a Singapore MOE Tier-2 grant MOE2019-T2-2-181.

REFERENCES

[1] Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of KDD*. Washington, DC, USA, August 24-27, 2003, 39–48.

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146.

[3] Ursin Brunner and Kurt Stockinger. 2020. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *Proceedings of EDBT*. Copenhagen, Denmark, March 30 - April 02, 2020, 463–473.

[4] Yixin Cao, Lei Hou, Juanzi Li, and Zhiyuan Liu. 2018. Neural Collective Entity Linking. In *Proceedings of COLING*. Santa Fe, New Mexico, USA, August 20-26, 2018, 675–686.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. Minneapolis, MN, USA, June 2-7, 2019, 4171–4186.

[6] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proc. VLDB Endow.* 11, 11 (2018), 1454–1467.

[7] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about Record Matching Rules. *Proc. VLDB Endow.* 2, 1 (2009), 407–418.

[8] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. 2020. Hierarchical Matching Network for Heterogeneous Entity Resolution. In *Proceedings of IJCAI*. Yokohama, Japan, January 7-15, 2021, 3665–3671.

[9] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-End Multi-Perspective Matching for Entity Resolution. In *Proceedings of IJCAI*. Macao, China, August 10-16, 2019, 4961–4967.

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

[11] Binbin Gu, Zhixu Li, Xiangliang Zhang, An Liu, Guanfeng Liu, Kai Zheng, Lei Zhao, and Xiaofang Zhou. 2017. The Interaction Between Schema Matching and Record Matching in Data Integration. *IEEE Trans. Knowl. Data Eng.* 29, 1 (2017), 186–199.

[12] Nitish Gupta, Sameer Singh, and Dan Roth. 2017. Entity Linking via Joint Encoding of Types, Descriptions, and Context. In *Proceedings of EMNLP*. Copenhagen, Denmark, September 9-11, 2017, 2681–2690.

[13] Xianpei Han, Le Sun, and Jun Zhao. 2011. Collective entity linking in web text: a graph-based method. In *Proceeding of SIGIR*. Beijing, China, July 25-29, 2011, 765–774.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of CVPR*. Las Vegas, NV, USA, June 27-30, 2016, 770–778.

[15] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. NAIS: Neural Attentive Item Similarity Model for Recommendation. *IEEE Trans. Knowl. Data Eng.* 30, 12 (2018), 2354–2366.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780.

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR (Poster)*. San Diego, CA, USA, May 7-9, 2015.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of ICLR (Poster)*. Toulon, France, April 24-26, 2017.

[19] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proc. VLDB Endow.* 9, 12 (2016), 1197–1208.

[20] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.* 3, 1 (2010), 484–493.

[21] Ioannis K. Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: Duplicate Detection with Matching Dependencies. *Proc. VLDB Endow.* 13, 5 (2020), 712–725.

[22] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. 2021. Improving the Efficiency and Effectiveness for BERT-based Entity Resolution. In *Proceedings of AAAI*. Virtual Event, February 2-9, 2021, 13226–13233.

[23] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks. In *Proceedings of AAAI*. New York, NY, USA, February 7-12, 2020, 8172–8179.

[24] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60.

[25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. (2019). arXiv:1907.11692

[26] Hanchao Ma, Morteza Alipour Langouri, Yinghui Wu, Fei Chiang, and Jiaxing Pi. 2019. Ontology-based Entity Matching in Attributed Graphs. *Proc. VLDB Endow.* 12, 10 (2019), 1195–1207.

[27] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. In *Proceedings of SIGMOD*. Online, USA, June 14-19, 2020, 1133–1147.

[28] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond. In *Proceedings of SIGMOD*. 1303–1316.

[29] 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 SIGMOD*. Houston, TX, USA, June 10-15, 2018, 19–34.

[30] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. Deep Sequence-to-Sequence Entity Matching for Heterogeneous Entity Resolution. In *Proceedings of CIKM*. Beijing, China, November 3-7, 2019, 629–638.

[31] Matteo Paganelli, Francesco Del Buono, Marco Pevarello, Francesco Guerra, and Maurizio Vincini. 2021. Automated Machine Learning for Entity Matching Tasks. In *Proceedings of EDBT*. Nicosia, Cyprus, March 23 - 26, 2021, 325–330.

[32] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of EMNLP*. Doha, Qatar,October 25-29, 2014, 1532–1543.

[33] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of NAACL-HLT*. New Orleans, Louisiana, USA, June 1-6, 2018, 2227–2237.

[34] Federico Piai, Donatella Firmani, Valter Crescenzi, Andrea De Angelis, Xin Luna Dong, Maurizio Mazzei, Paolo Merialdo, and Divesh Srivastava (Eds.). 2020. *Proceedings of DI2KG@Proc. VLDB Endow.* Tokyo, Japan, August 31, 2020.

[35] Anna Primpeli, Ralph Peeters, and Christian Bizer. 2019. The WDC Training Dataset and Gold Standard for Large-Scale Product Matching. In *Proceedings of WWW*. San Francisco, CA, USA, May 13-17, 2019, 381–386.

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

[37] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Generating Concise Entity Matching Rules. In *Proceedings of SIGMOD*. Chicago, IL, USA, May 14-19, 2017, 1635–1638.

[38] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing Entity Matching Rules by Examples. *Proc. VLDB Endow.* 11, 2 (2017), 189–202.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Proceedings of NIPS*. Long Beach, CA, USA, December 4-9, 2017, 5998–6008.

[40] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proceedings of ICLR (Poster)*. Vancouver, BC, Canada, April 30 - May 3, 2018.

[41] Danqing Wang, Pengfei Liu, Yining Zheng, Xipeng Qiu, and Xuan-Jing Huang. 2020. Heterogeneous Graph Neural Networks for Extractive Document Summarization. In *Proceedings of ACL*. Online, July 5-10, 2020, 6209–6219.

[42] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. *Proc. VLDB Endow.* 5, 11 (2012), 1483–1494.

[43] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. 2011. Entity Matching: How Similar Is Similar. *Proc. VLDB Endow.* 4, 10 (2011), 622–633.

[44] Menghan Wang, Yujie Lin, Guli Lin, Keping Yang, and Xiao-Ming Wu. 2020. M2GRL: A Multi-task Multi-view Graph Representation Learning Framework for Web-scale Recommender Systems. In *Proceedings of KDD*. Virtual Event, USA, August 23-27, 2020, 2349–2358.

[45] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *Proceedings of KDD*. Anchorage, AK, USA, August 4-8, 2019, 950–958.

[46] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. (2019). arXiv:1910.03771

[47] Ledell Yu Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. StarSpace: Embed All The Things!. In *Proceedings of AAAI*. New Orleans, Louisiana, USA, February 2-7, 2018, 5569–5577.

[48] Fan Xiong and Jianliang Gao. 2019. Entity Alignment for Cross-lingual Knowledge Graph with Graph Convolutional Networks. In *Proceedings of IJCAI*. Macao, China, August 10-16, 2019, 6480–6481.

[49] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and

Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of ICML*. Lille, France, 6-11 July 2015, 2048–2057.

[50] Mengge Xue, Weiming Cai, Jinsong Su, Linfeng Song, Yubin Ge, Yubao Liu, and Bin Wang. 2019. Neural Collective Entity Linking Based on Recurrent Random Walk Network Learning. In *Proceedings of IJCAI*. Macao, China, August 10-16, 2019, 5327–5333.

[51] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Proceedings of NeurIPS*. Vancouver, BC, Canada, December 8-14, 2019, 5754–5764.

[52] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of AAAI*. Honolulu, Hawaii, USA, January 27 - February 1, 2019, 7370–7377.

[53] Dongxiang Zhang, Long Guo, Xiangnan He, Jie Shao, Sai Wu, and Heng Tao Shen. 2018. A Graph-Theoretic Fusion Framework for Unsupervised Entity Resolution. In *Proceedings of ICDE*. Paris, France, April 16-19, 2018, 713–724.

[54] Dongxiang Zhang, Yuyang Nie, Sai Wu, Yanyan Shen, and Kian-Lee Tan. 2020. Multi-Context Attention for Entity Matching. In *Proceedings of WWW*. Taipei, Taiwan, April 20-24, 2020, 2634–2640.

[55] Fanjin Zhang, Xiao Liu, Jie Tang, Yuxiao Dong, Peiran Yao, Jie Zhang, Xiaotao Gu, Yan Wang, Bin Shao, Rui Li, and Kuansan Wang. 2019. OAG: Toward Linking Large-scale Heterogeneous Entity Graphs. In *Proceedings of KDD*. Anchorage, AK, USA, August 4-8, 2019, 2585–2595.

[56] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. 2019. Multi-view Knowledge Graph Embedding for Entity Alignment. In *Proceedings of IJCAI*. Macao, China, August 10-16, 2019, 5429–5435.