Deep Entity Matching with Pre-Trained Language Models [Scalable Data Science]

Yuliang Li[1], Jinfeng Li[1], Yoshihiko Suhara[1], AnHai Doan[2], and Wang-Chiew Tan[1]

[1]Megagon Labs [2]University of Wisconsin-Madison
[1]{yuliang, jinfeng, yoshi, wangchiew}@megagon.ai, [2]anhai@cs.wisc.edu

ABSTRACT

We present Ditto, a novel entity matching system based on pre-trained Transformer-based language models. We fine-tune and cast EM as a sequence-pair classification problem to leverage such models with a simple architecture. Our experiments show that a straightforward application of language models such as BERT, DistilBERT, or ALBERT pre-trained on large text corpora already significantly improves the matching quality and outperforms previous state-of-the-art (SOTA), by up to 19% of F1 score on benchmark datasets. We also developed three optimization techniques to further improve Ditto's matching capability. Ditto allows domain knowledge to be injected by highlighting important pieces of input information that may be of interest when making matching decisions. Ditto also summarizes strings that are too long so that only the essential information is retained and used for EM. Finally, Ditto adapts a SOTA technique on data augmentation for text to EM to augment the training data with (difficult) examples. This way, Ditto is forced to learn "harder" to improve the model's matching capability. The optimizations we developed further boost the performance of Ditto by up to 8.5%. Perhaps more surprisingly, we establish that Ditto can achieve the previous SOTA results with at most half the number of labeled data. Finally, we demonstrate Ditto's effectiveness on a real-world large-scale EM task. On matching two company datasets consisting of 789K and 412K records, Ditto achieves a high F1 score of 96.5%.

PVLDB Reference Format:
Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, Wang-Chiew Tan. Deep Entity Matching with Pre-Trained Language Models. *PVLDB*, (): xxxx-yyyy, .
DOI:

1. INTRODUCTION

Entity Matching (EM) refers to the problem of determining whether two data entries refer to the same real-world entity. Consider the two datasets about products in Figure 1. The goal is to determine the set of pairs of data entries, one entry from each table so that each pair of entries refer to the same product.

Proceedings of the VLDB Endowment, Vol. , No.
ISSN 2150-8097.
DOI:

If the datasets are large, it can be expensive to determine the pairs of matching entries. For this reason, EM is typically accompanied by a pre-processing step, called *blocking*, to prune pairs of entries that are unlikely matches to reduce the number of candidate pairs to consider. As we will illustrate, correctly *matching* the candidate pairs requires substantial language understanding and domain-specific knowledge. Hence, entity matching remains a challenging task even for the most advanced EM solutions.

We present Ditto, a novel EM solution based on pre-trained Transformer-based language models (or *pre-trained language models* in short). We cast EM as a sequence-pair classification problem to leverage such models, which have been shown to generate highly contextualized embeddings that capture better language understanding compared to traditional word embeddings. Ditto further improves its matching capability through three optimizations: (1) It allows domain knowledge to be added by highlighting important pieces of the input that may be useful for matching decisions. (2) It summarizes long strings so that only the most essential information is retained and used for EM. (3) It augments training data with (difficult) examples, which challenges Ditto to learn "harder" and also reduces the amount of training data required. Figure 2 depicts Ditto in the overall architecture of a complete EM workflow.

There are 9 candidate pairs of entries to consider for matching in total in Figure 1. The blocking heuristic that matching entries must have one word in common in the **title** will reduce the number of pairs to only 3: the first entry on the left with the first entry on the right and so on. Perhaps more surprisingly, even though the 3 pairs are highly similar and look like matches, only the first and last pair of entries are true matches. Our system, Ditto, is able to discern the nuances in the 3 pairs to make the correct conclusion for every pair while some state-of-the-art systems are unable to do so.

The example illustrates the power of language understanding given by Ditto's pre-trained language model. It understands that *instant immersion spanish deluxe 2.0* is the same as *instant immers spanish dlux 2* in the context of software products even though they are syntactically different. Furthermore, one can explicitly emphasize that certain parts of a value are more useful for deciding matching decisions. For books, the domain knowledge that the grade level or edition is important for matching books can be made explicit to Ditto, simply by placing tags around the grade/edition values. Hence, for the second candidate pair, even though the titles are highly similar (i.e., they overlap in many words), Ditto is able to focus on the grade/edition information when making the matching decision. The third candidate pair shows the power of language understanding for the opposite situation. Even though the entries look dissimilar Ditto is able to attend to the right parts of a value (i.e., the manf./modelno under different attributes) and also understand the semantics of the model number to make the right decision.

title	manf./modelno	price
instant immersion spanish deluxe 2.0	topics entertainment	49.99
adventure workshop 4th-6th grade 7th edition	encore software	19.99
sharp printing calculator	sharp el1192bl	37.63

title	manf./modelno	price
instant immers spanish dlux 2	NULL	36.11
encore inc adventure workshop 4th-6th grade 8th edition	NULL	17.1
new-sharp shr-el1192bl two-color printing calculator 12-digit lcd black red	NULL	56.0

Figure 1: Entity Matching: determine the matching entries from two datasets.

Contributions In summary, the following are our contributions:

- We present DITTO, a novel EM solution based on pre-trained language models (LMs) such as BERT, DistilBERT, and ALBERT. We fine-tune and cast EM as a sequence-pair classification problem to leverage such models with a simple architecture. To the best of our knowledge, DITTO is the first EM solution that leverages pre-trained Transformer-based LMs, which are powerful LMs that have been shown to provide deeper language understanding.
- We also developed three optimization techniques to further improve DITTO's matching capability through injecting domain knowledge, summarizing long strings, and augmenting training data with (difficult) examples. The first two techniques help DITTO focus on the right information for making matching decisions. The last technique, data augmentation, is adapted from [24] for EM to help DITTO learn "harder" to understand the data invariance properties that may exist but are beyond the provided labeled examples and also, reduce the amount of training data required.
- We evaluated the effectiveness of DITTO on three benchmark datasets: the Entity Resolution benchmark [21], the Magellan dataset [20], and the WDC product matching dataset [31] of various sizes and domains. Our experimental results show that DITTO consistently outperforms the previous SOTA EM solutions in all datasets and by up to 25% in F1 scores. Furthermore, DITTO consistently performs better on dirty data and is more label efficient: it achieves the same or higher previous SOTA accuracy using less than half the labeled data.
- We applied DITTO to a real-world large-scale matching task on two company datasets, containing 789K and 412K entries respectively. To deploy an end-to-to EM pipeline efficiently, we developed an advanced blocking technique to help reduce the number of pairs to consider for DITTO. DITTO obtains high accuracy, 96.5% F1 on a holdout dataset. The blocking phase also helped speed up the end-to-end EM deployment significantly, by up to 3.8 times, compared to naive blocking techniques.
- Finally, we will open-source DITTO in the future.

Outline Section 2 overviews DITTO and pre-trained LMs. Section 3 describes how we optimize DITTO with domain knowledge, summarization, and data augmentation. Our experimental results are described in Section 4 and the case study is presented in Section 5. We discuss related work in Section 6 and conclude in Section 7.

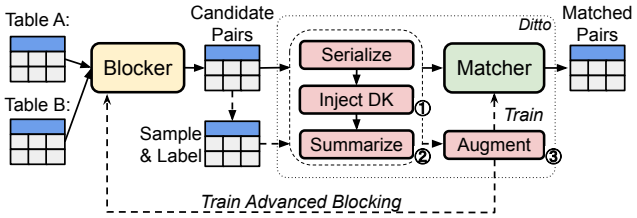

Figure 2: An EM system architecture with DITTO as the matcher. In addition to the training data, the user of DITTO can specify (1) a method for injecting domain knowledge (DK), (2) a summarization module for keeping the essential information, and (3) a data augmentation (DA) operator to strengthen the training set.

2. BACKGROUND AND ARCHITECTURE

We present the main concepts behind EM and provide some background on pre-trained LMs before we describe how we fine-tune the LMs on EM datasets to train EM models. We also present a simple method for reducing EM to a sequence-pair classification problem so that pre-trained LMs can be used for solving the EM problem.

Notations DITTO's EM pipeline takes as input two collections D and D' of data entries (e.g., rows of relational tables, XML documents, JSON files, text paragraphs) and outputs a set $M \subseteq D \times D'$ of pairs where each pair $(e, e') \in M$ is thought to represent the same real-world entity (e.g., person, company, laptop, etc.). A data entry e is a set of key-value pairs $e = \{(\texttt{attr}_i, \texttt{val}_i)\}_{1 \leq i \leq k}$ where $\texttt{attr}_i$ is the attribute name and $\texttt{val}_i$ is the attribute's value represented as text. Note that our definition of data entries is general enough to capture both structured and semi-structured data such as JSON files.

As described earlier, an end-to-end EM system consists of a *blocker* and a *matcher*. The goal of the blocking phase is to quickly identify a small subset of $D \times D'$ of candidate pairs of high recall (i.e., a high proportion of actual matching pairs are that subset). The goal of a matcher (i.e., DITTO) is to accurately predict, given a pair of entries, whether they refer to the same real-world entity.

2.1 Pre-trained language models

Unlike prior learning-based EM solutions that rely on word embeddings and customized RNN architectures to train the matching model (See Section 6 for a detailed summary), DITTO trains the matching models by fine-tuning pre-trained LMs in a simpler architecture.

Pre-trained LMs such as BERT [11], ALBERT [22], and GPT-2 [32] have demonstrated good performance on a wide range of NLP tasks. They are typically deep neural networks with multiple Transformer layers [42], typically 12 or 24 layers, pre-trained on large text corpora such as Wikipedia articles in an unsupervised manner. During pre-training, the model is self-trained to perform auxiliary tasks such as missing token and next-sentence prediction. Studies [7, 41] have shown that the shallow layers capture lexical meaning while the deeper layers capture syntactic and semantic meanings of the input sequence after pre-training.

A specific strength of pre-trained LMs is that it learns the semantics of words better than conventional word embedding techniques such as word2vec, GloVe, or FastText. This is largely because the Transformer architecture calculates token embeddings from all the tokens in the input sequence and thus, the embeddings it generates are *highly-contextualized* and captures the semantic and contextual understanding of the words. Consequently, such embeddings can capture polysemy, i.e., discern that the same word may have different meanings in different phrases. For example, the word *Sharp* has different meanings in *"Sharp resolution"* versus *"Sharp TV"*. Pre-trained LMs will embed "*Sharp*" differently depending on the context while traditional word embedding techniques such as FastText always produce the same vector independent of the context. Such models can also understand the opposite, i.e., that different words may have the same meaning. For example, the words *immersion* and *immers* (respectively, (*deluxe*, *dlux*) and (2.0, 2)) are likely the same

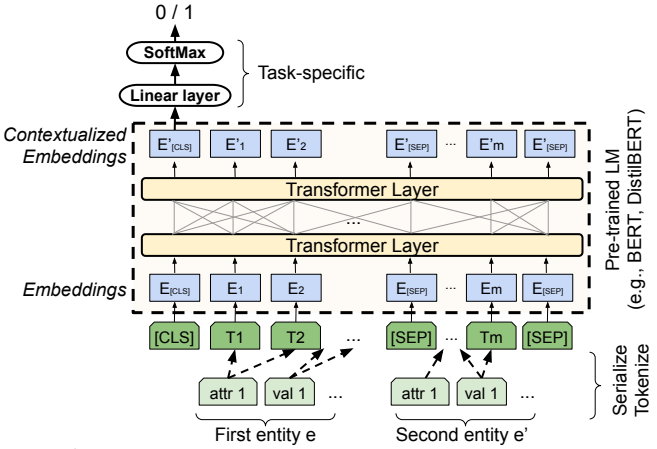

Figure 3: DITTO's model architecture. DITTO serializes the two entries as one sequence and feeds it to the model as input. The model consists of (1) token embeddings and Transformer layers [49] from a pre-trained language model (e.g., BERT) and (2) task-specific layers (linear followed by softmax). Conceptually, the [CLS] token "summarizes" all the contextual information needed for matching as a contextualized embedding vector $E'_{[CLS]}$ which the task-specific layers take as input for classification.

given their respective contexts. Thus, such language understanding capability of pre-trained LMs can improve the EM performance.

2.2 Fine-tuning pre-trained language models

A pre-trained LM can be fine-tuned with task-specific training data so that it becomes better at performing that task. Here, we fine-tune a pre-trained LM for the EM task with a labeled training dataset consisting of positive and negative pairs of matching and non-matching entries as follows:

1. Add task-specific layers after the final layer of the LM. For EM, we add a simple fully connected layer and a softmax output layer for binary classification.
2. Initialize the modified network with parameters from the pre-trained LM.
3. Train the modified network on the training set until it converges.

The result is a model fine-tuned for the EM task. In DITTO, we fine-tune the popular base 12-layer BERT model [11] and its distilled variant DistilBERT [36], which is smaller but more efficient. However, our proposed techniques are independent of the choice of pre-trained LMs and our experimental results (Table 6) indicate that DITTO can potentially perform even better with larger pre-trained LMs. We illustrate the model architecture in Figure 3. The pair of data entries is serialized (see next section) as input to the LM and the output is a match or no-match decision. DITTO's architecture is much simpler when compared to many state-of-the-art EM solutions today [27, 12]. Even though the bulk of the "work" is simply off-loaded to pre-trained LMs, we show that this simple scheme works surprisingly well in our experiments.

2.3 Serializing the data entries for Ditto

Since LMs take token sequences (i.e., text) as input, a key challenge is to convert the candidate pairs into token sequences so that they can be meaningfully ingested by DITTO.

DITTO serializes data entries as follows: for each data entry $e = \{(\mathsf{attr}_i, \mathsf{val}_i)\}_{1 \le i \le k}$, we let

$$\mathsf{serialize}(e) ::= [\mathsf{COL}]\ \mathsf{attr}_1\ [\mathsf{VAL}]\ \mathsf{val}_1 \ldots [\mathsf{COL}]\ \mathsf{attr}_k\ [\mathsf{VAL}]\ \mathsf{val}_k,$$

where [COL] and [VAL] are special tokens for indicating the start of attribute names and values respectively. For example,the first entry of the second table is serialized as:

[COL] title [VAL] instant immers spanish dlux 2 [COL] manf./modelno [VAL] NULL [COL] price [VAL] 36.11

To serialize a candidate pair (e, e'), we let

$$\mathsf{serialize}(e, e') ::= [\mathsf{CLS}]\ \mathsf{serialize}(e)\ [\mathsf{SEP}]\ \mathsf{serialize}(e')\ [\mathsf{SEP}],$$

where [SEP] is the special token separating the two sequences and [CLS] is the special token necessary for BERT to encode the sequence pair into a 768-dimensional vector which will be fed into the fully connected layers for classification.

Other serialization schemes There are different ways to serialize data entries so that LMs can treat the input as a sequence classification problem. For example, one can also omit the special tokens "[COL]" and/or "[VAL]", or exclude attribute names attr_i during serialization. We found that including the special tokens to retain the structure of the input does not hurt the performance in general and excluding the attribute names tend to help only when the attribute names do not contain useful information (e.g., names such as attr1, attr2, ...) or when the entries contain only one column. A more rigorous study on this matter is left for future work.

Heterogeneous schemas As shown, the serialization method of DITTO does not require data entries to adhere to the same schema. It also does not require that the attributes of data entries to be matched prior to executing the matcher, which is a sharp contrast to other EM systems such as DeepER [12] or DeepMatcher[1] [27]. Furthermore, DITTO can also ingest and match hierarchically structured data entries by serializing nested attribute-value pairs with special start and end tokens (much like Lisp or XML-style parentheses structure).

3. OPTIMIZATIONS IN DITTO

As we will describe in Section 4, the basic version of DITTO, which leverages only the pre-trained LM, is already outperforming the SOTA on average. Here, we describe three further optimization techniques that will facilitate and challenge DITTO to learn "harder", and consequently make better matching decisions.

3.1 Leveraging Domain Knowledge

Our first optimization allows domain knowledge to be injected into DITTO through *pre-processing* the input sequences (i.e., serialized data entries) to emphasize what pieces of information are potentially important. This follows the intuition that when human workers make a matching/non-matching decision on two data entries, they typically look for spans of text that contain key information before making the final decision. Even though we can also train deep learning EM solutions to learn such knowledge, we will require a significant amount of training data to do so. As we will describe, this pre-processing step on the input sequences is lightweight and yet can yield significant improvements. Our experiment results show that with less than 5% of additional training time, we can improve the model's performance by up to 8%.

There are two main types of domain knowledge that we can provide DITTO.

Span Typing The type of a span of tokens is one kind of domain knowledge that can be provided to DITTO. Product id, street number, publisher are examples of span types. Span types help DITTO avoid mismatches. With span types, for example, DITTO is likelier to avoid matching a street number with a year or a product id.

Table 1 summarizes the main span types that human workers would focus on when matching three types of entities in our benchmark datasets.

[1] In DeepMatcher, the requirement that both entries have the same schema can be removed by treating the values in all columns as one value under one attribute.

Table 1: Main span types for matching entities in our benchmark datasets.

Entity Type	Types of Important Spans
Publications, Movies, Music	Persons (e.g., Authors), Year, Publisher
Organizations, Employers	Last 4-digit of phone, Street number
Products	Product ID, Brand, Configurations (num.)

The developer specifies a recognizer to type spans of tokens from attribute values. The recognizer takes a text string v as input and returns a list $\mathsf{recognizer}(v) = \{(s_i, t_i, \mathsf{type}_i)\}_{i \geq 1}$ of start/end positions of the span in v and the corresponding type of the span. Ditto's current implementation leverages an open-source Named-Entity Recognition (NER) model [39] to identify known types such as persons, dates, or organizations and use regular expressions to identify specific types such as product IDs, last 4 digits of phone numbers, etc.

After the types are recognized, the original text v is replaced by a new text where special tokens are inserted to reflect the types of the spans. For example, a phone number "(866) 246-6453" may be replaced with "(866) 246 - [LAST] 6453 [/LAST]" where [LAST]/[/LAST] indicates the start/end of the last 4 digits and additional spaces are also added because of tokenization. In our implementation, when we are sure that the span type has only one token or the NER model is inaccurate in determining the end position, we drop the end indicator and keep only the start indicator token.

Intuitively, these newly added special tokens are additional signals to the self-attention mechanism that already exists in pre-trained LMs, such as BERT. If two spans have the same type, then Ditto picks up the signal that they are likelier to be the same and hence, they are aligned together for matching. In the above example,

"..246- [LAST] 6453 [/LAST] .. [SEP] .. [LAST] 0000 [/LAST].."

when the model sees two encoded sequences with the [LAST] special tokens, it is likely to take the hint to align "6453" with "0000" without relying on other patterns elsewhere in the sequence that may be harder to learn.

Span Normalization The second kind of domain knowledge that can be passed to Ditto rewrites syntactically different but equivalent spans into the same string. This way, they will have identical embeddings and it becomes easier for Ditto to detect that the two spans are identical. For example, we can enforce that "VLDB journal" and "VLDBJ" are the same by writing them as VLDBJ. Similarly, we can enforce the general knowledge that "5 %" vs. "5.00 %" are equal by writing them as "5.0%".

The developer specifies a set of rewriting rules to rewrite spans. The specification consists of a function that first identifies the spans of interest before it replaces them with the rewritten spans. Ditto contains a number of rewriting rules for numbers, including rules that round all floating point numbers to 2 decimal places and dropping all commas from integers (e.g., "2,020" $\rightarrow$ "2020"). For abbreviations, we allow the developers to specify a dictionary of synonym pairs to normalize all synonym spans to be the same.

3.2 Summarizing long entries

When the value is an extremely long string, it becomes harder for the LM to understand what to pay attention to when matching. In addition, one limiting factor of Transformer-based pre-trained LMs is that there is a limit on the sequence length of the input. For example, the input to BERT can have at most 512 sub-word tokens. It is thus important to summarize the serialized entries down to the maximum allowed length while retaining the key information. A common practice is to truncate the sequences so that they fit within the maximum length. However, the truncation strategy does not work well for EM in general because the important information for matching is usually not at the beginning of the sequences.

There are many ways to perform summarization [25, 33, 35]. In Ditto's current implementation, we use a TF-IDF-based summarization technique that *retains non-stopword tokens with the high TF-IDF scores*. We ignore the start and end tags generated by span typing in this process and use the list of stop words from scikit-learn library. By doing so, Ditto feeds only the most informative tokens to the LM. We found that this technique works well in practice. Our experiment results show that it improves the F1 score of Ditto on a text-heavy dataset from $\sim$40% to over 92% and we plan to add more summarization techniques to Ditto's library in the future.

3.3 Augmenting training data

We describe how we apply data augmentation to augment the training data for entity matching.

Data augmentation (DA) is a commonly used technique in computer vision for generating additional training data from existing examples by simple transformation such as cropping, flipping, rotation, padding, etc. The DA operators not only add more training data, but the augmented data also allows to model to learn to make predictions invariant of these transformations.

Similarly, DA can add training data that will help EM models learn "harder". Although labeled examples for EM are arguably not hard to obtain, invariance properties are very important to help make the solution more robust to dirty data, such as missing values (NULLs), values that are placed under the wrong attributes or missing some tokens.

Next, we introduce a set of DA operators for EM that will help train more robust models.

Augmentation operators for EM The proposed DA operators are summarized in Table 2. If s is a serialized pair of data entries with a match or no-match label l, then an augmented example is a pair (s', l), where s' is obtained by applying an operator o on s and s' has the same label l as before.

Table 2: Data augmentation operators in Ditto. The operators are 3 different levels: span-level, attribute-level, and entry-level. All samplings are done uniformly at random.

Operator	Explanation
span_del	Delete a randomly sampled span of tokens
span_shuffle	Randomly sample a span and shuffle the tokens' order
attr_del	Delete a randomly chosen attribute and its value
attr_shuffle	Randomly shuffle the orders of all attributes
entry_swap	Swap the order of the two data entries e and e'

The operators are divided into 3 categories. The first category consists of span-level operators, such as span_del and span_shuffle. These two operators are used in NLP tasks [48, 24] and shown to be effective for text classification. For span_del, we randomly delete from s a span of tokens of length at most 4 without special tokens (e.g., [SEP], [COL], [VAL]). For span_shuffle, we sample a span of length at most 4 and randomly shuffle the order of its tokens.

These two operators are motivated by the observation that making a match/no-match decision can sometimes be "too easy" when the candidate pair of data entries contain multiple spans of text supporting the decision. For example, suppose our negative examples for matching company data in the existing training data is similar to what is shown below.

[CLS] ... [VAL] Google LLC ... [VAL] (866) 246-6453 [SEP] ...
[VAL] Alphabet inc ... [VAL] (650) 253-0000 [SEP]

The model may learn to predict "no-match" based on the phone number alone, which is insufficient in general. On the other hand,

by corrupting parts of the input sequence (e.g., dropping phone numbers), DA forces the model to learn beyond that, by leveraging the remaining signals, such as the company name, to predict "no-match".

The second category of operators is attribute-level operators: attr_del and attr_shuffle. The operator attr_del randomly deletes an attribute (both name and value) and attr_shuffle randomly shuffles the order of the attributes of both data entries. The motivation for attr_del is similar to span_del and span_shuffle but it gets rid of an attribute entirely. The attr_shuffle operator allows the model to learn the property that the matching decision should be independent of the ordering of attributes in the sequence.

The last operator, entry_swap, swaps the order of the pair (e, e') with probability $1/2$. This teaches the model to make symmetric decisions (i.e., $F(e, e') = F(e', e)$) and helps double the size of the training set if both input tables are from the same data source.

MixDA: interpolating the augmented data Unlike DA operators for images which almost always preserve the image labels, the operators for EM can distort the input sequence so much that the label becomes incorrect. For example, the attr_del operator may drop the company name entirely and the remaining attributes may contain no useful signals to distinguish the two entries.

To address this issue, DITTO applies MixDA, a recently proposed data augmentation technique for NLP tasks [24] illustrated in Figure 4. Instead of using the augmented example directly, MixDA computes a convex interpolation of the original example with the augmented examples. Hence, the interpolated example is somewhere in between, i.e., it is a "partial" augmentation of the original example and this interpolated example is expected to be less distorted than the augmented one.

The idea of interpolating two examples is originally proposed for computer vision tasks [53]. For EM or text data, since we cannot directly interpolate sequences, MixDA interpolates their representations by the language model instead. We omit the technical details and refer the interested readers to [24]. In practice, augmentation with MixDA slows the training time because the LM is called twice. However, the prediction time is not affected since the DA operators are only applied to training data.

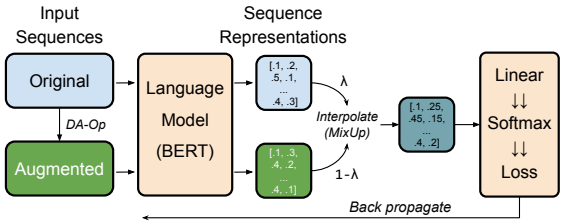

Figure 4: Data augmentation with MixDA. To apply MixDA, we first transform the example with a DA operator and pass it to the LM. We then interpolate the representations of the original and the augmented examples. Finally, we feed the interpolation to the rest of the NN and back-propagate.

4. EXPERIMENTS

We present the experiment results on benchmark datasets for EM: the ER Benchmark datasets [21], the Magellan datasets [20] and the WDC product data corpus [31]. DITTO achieves new SOTA results on all these datasets and outperforms the previous best results by up to 25% in F1 score. The results show that DITTO is more robust to dirty data and performs well when the training set is small. DITTO is also more label-efficient as it achieves the previous SOTA results using only 1/2 of the training data across multiple subsets of the WDC corpus. Our ablation analysis shows that (1) using pre-trained LMs contributes to over 50% of DITTO's performance gain and (2) all 3 optimizations, domain knowledge (DK), summarization (SU) and data augmentation (DA), are effective. For example, SU improves the performance on a text-heavy dataset by 41%, DK leads to 1.98% average improvement on the ER-Magellan datasets and DA improves on the WDC datasets by 2.53% on average.

Table 3: The 13 datasets divided into 4 categories of domains. The datasets marked with † are text-heavy (Textual). Each dataset with ∗ has an additional dirty version to test the models' robustness against noisy data.

Datasets	Domains
Amazon-Google, Walmart-Amazon*	software / electronics
Abt-Buy†, Beer	product
DBLP-ACM*, DBLP-Scholar*, iTunes-Amazon*	citation / music
Company†, Fodors-Zagats	company / restaurant

4.1 Benchmark datasets

We experimented with all the 13 publicly available datasets used for evaluating DeepMatcher [27]. These datasets are from the ER Benchmark datasets [21] and the Magellan data repository [10]. We summarize the datasets in Table 3 and refer to them as ER-Magellan. These datasets are for training and evaluating matching models for various domains including products, publications, and businesses. Each dataset consists of candidate pairs from two structured tables of entity records of the same schema. The pairs are sampled from the results of blocking and manually labeled. The positive rate (i.e., the ratio of matched pairs) ranges from 9.4% (Walmart-Amazon) to 25% (Company). The number of attributes ranges from 1 to 8.

Among the datasets, the Abt-Buy and Company datasets are text-heavy meaning that at least one attributes contain long text. Also, following [27], we use the dirty version of the DBLP-ACM, DBLP-Scholar, iTunes-Amazon, and Walmart-Amazon datasets to measure the robustness of the models against noise. These datasets are generated from the clean version by randomly emptying attributes and appending their values to another randomly selected attribute.

Each dataset is split into the training, validation, and test sets using the ratio of 3:1:1. We list the size of each dataset in Table 5.

The WDC product data corpus [31] contains 26 million product offers and descriptions collected from e-commerce websites [47]. The goal is to find product offer pairs that refer to the same product. To evaluate the accuracy of product matchers, the dataset provides 4,400 manually created golden labels of offer pairs from 4 categories: computers, cameras, watches, and shoes. Each category has a fixed number of 300 positive and 800 negative pairs. For training, the dataset provides for each category pairs that share the same product ID such as GTINs or MPNs mined from the product's webpage. The negative examples are created by selecting pairs that have high textual similarity but different IDs. These labels are further reduced to different sizes to test the models' label efficiency. We summarize the different subsets in Table 4. We refer to these subsets as the WDC datasets.

Table 4: Different subsets of the WDC product data corpus. Each subset (except Test) is split into a training set and a validation set with a ratio of 4:1. The last column shows the positive rate (%POS) of each category in the xLarge set. The positive rate on the test set is 27.27% for all the categories.

Categories	Test	Small	Medium	Large	xLarge	%POS
Computers	1,100	2,834	8,094	33,359	68,461	14.15%
Cameras	1,100	1,886	5,255	20,036	42,277	16.98%
Watches	1,100	2,255	6,413	27,027	61,569	15.05%
Shoes	1,100	2,063	5,805	22,989	42,429	9.76%
All	4,400	9,038	25,567	103,411	214,736	14.10%

Each entry in this dataset has 5 attributes. DITTO uses only the title attribute because it contains rich product information such as brands, IDs, and price, making the rest of the attributes redundant.

Meanwhile, DeepMatcher is allowed to use any subsets of attributes to determine the best attribute set as in [31].

4.2 Implementation and experimental setup

We implemented Ditto in PyTorch [29] and the Transformers library [49]. The default setting uses the uncased 6-layer DistilBERT [36] pre-trained model and half-precision floating-point (fp16) to accelerate the training and prediction speed. In all the experiments, we fix the learning rate to be 3e-5 and the max sequence length to be 256. The batch size is 32 if MixDA is used and 64 otherwise. The training process runs a fixed number of epochs (10, 15, or 40 depending on the dataset size) and returns the checkpoint with the highest F1 score on the validation set. We conducted all experiments on a p3.8xlarge AWS EC2 machine with 4 V100 GPUs (one GPU per run).

Compared methods. We compare Ditto with the SOTA EM solution DeepMatcher and its variants. We also compare with variants of Ditto without the data augmentation (DA) and/or domain knowledge (DK) optimization to evaluate the effectiveness of each component. We summarize these methods below. We report the average F1 of 5 repeated runs in all the settings.

- **DeepMatcher:** DeepMatcher [27] is the SOTA matching solution. Compared to Ditto, DeepMatcher customizes the RNN architecture to aggregate the attribute values, then compares/aligns the aggregated representations of the attributes. DeepMatcher leverages FastText [4] to train the word embeddings. When reporting DeepMatcher's F1 scores, we use the numbers in [27] for the ER-Magellan datasets and numbers in [31] for the WDC datasets. We also reproduced those results using the open-sourced implementation and report the training time.
- **DeepMatcher+:** Follow-up work [19] slightly outperforms DeepMatcher in the DBLP-ACM dataset and [15] achieves better F1 in the Walmart-Amazon and Amazon-Google datasets. According to [27], the Magellan system ([20], based on classical ML models) outperforms DeepMatcher in the Beer and iTunes-Amazon datasets. For these cases, we denote by DeepMatcher+ the best F1 scores among DeepMatcher and these works aforementioned.
- **Ditto:** This is the full version of our system with all 3 optimizations, domain knowledge (DK), TF-IDF summarization (SU), and data augmentation (DA) turned on. See the details below.
- **Ditto(DA):** This version only turns on the DA (with MixDA) and SU but does not have the DK optimization. We apply one of the span-level or attribute-level DA operators listed in Table 2 with the entry_swap operator. We compare the different combinations and report the best one. Following [24], we apply MixDA with the interpolation parameter λ sampled from a Beta distribution Beta$(0.8, 0.8)$.
- **Ditto(DK):** With only the DK and SU optimizations on, this version of Ditto is expected to have lower F1 scores but train much faster. We apply the span-typing to datasets of each domain according to Table 1 and apply the span-normalization on the number spans.
- **Baseline:** This base form of Ditto corresponds simply to fine-tuning a pre-trained LM (DistilBERT) on the EM task. We did not apply any optimizations on the baseline. We pick DistilBERT instead of larger models such as BERT or ALBERT because DistilBERT is faster to train and it also makes a tougher comparison for Ditto since larger models are generally perceived to have more powerful language understanding capabilities [52, 23, 22].

4.3 Main results

Table 5 shows the results of the ER-Magellan datasets. Overall, Ditto (with optimizations) achieves significantly higher F1 scores than the SOTA results (DeepMatcher+). Ditto without optimizations (i.e., the baseline) achieves comparable results with DeepMatcher+. Ditto outperforms DeepMatcher+ in 10/13 cases and by up to 25% (Dirty, Walmart-Amazon) while the baseline outperforms DeepMatcher+ in 8/13 cases and by up to 16% (Dirty, Walmart-Amazon). On the 3 cases that Ditto performs slightly worse than DeepMatcher+, it turns out that using a larger pre-trained LMs such as BERT or ALBERT helps fill the gaps (see Table 6). These initial results led us to believe that larger pre-trained language models will further improve Ditto's results and we leave as future work to further verify this hypothesis.

In addition, we found that Ditto is better at datasets with small training sets. Particularly, the average improvement on the 7 smallest datasets is 9.96% vs. 0.32% on average on the rest of datasets. Ditto is also more robust against data noise than DeepMatcher+. In the 4 dirty datasets, the performance degradation of Ditto is only 0.68 on average while the performance of DeepMatcher+ degrades by 8.21. These two properties make Ditto more attractive in practical EM settings.

Ditto also achieves promising results on the WDC datasets (Table 7). Ditto achieves the highest F1 score of 94.08 when using all the 215k training data, outperforming the previous best result by 3.92. Similar to what we found in the ER-Magellan datasets, the improvements are higher on settings with fewer training examples (to the right of Table 7). The results also show that Ditto is more *label efficient* than DeepMatcher. For example, when using only 1/2 of the data (Large), Ditto already outperforms DeepMatcher with all the training data (xLarge) by 2.89 in All. When using only 1/8 of the data (Medium), the performance is within 1% close to DeepMatcher's F1 when 1/2 of the data (Large) is in use. The only exception is the shoes category. This may be caused by the large gap of the positive label ratios between the training set and the test set (9.76% vs. 27.27% according to Table 4).

Table 5: F1 scores on the ER-Magellan EM datasets. The numbers of DeepMatcher+ (DM+) are the highest available found in [15, 19, 27].

Datasets	DM+	Ditto	Ditto (DA)	Ditto (DK)	Baseline	Size
Structured						
Amazon-Google	70.7	71.42 (+0.72)	72.10	71.53	70.04	11,460
Beer	78.8	82.12 (+3.32)	80.72	83.11	73.44	450
DBLP-ACM	98.45	98.65 (+0.2)	98.83	98.54	98.65	12,363
DBLP-Google	94.7	94.57 (-0.13)	94.53	94.53	94.67	28,707
Fodors-Zagats	100	97.76 (-2.24)	98.18	98.16	96.76	946
iTunes-Amazon	91.2	94.19 (+2.99)	90.23	92.47	91.38	539
Walmart-Amazon	73.6	80.66 (+7.06)	81.30	79.07	76.95	10,242
Dirty						
DBLP-ACM	98.1	98.63 (+0.53)	98.43	98.52	98.60	12,363
DBLP-Google	93.8	94.68 (+0.88)	94.48	94.62	94.66	28,707
iTunes-Amazon	79.4	93.16 (+13.76)	92.30	91.72	89.88	539
Walmart-Amazon	53.8	78.87 (+25.07)	78.80	76.79	70.40	10,242
Textual						
Abt-Buy	62.8	82.60 (+19.80)	81.66	81.90	81.74	9,575
Company	92.7	92.43 (-0.27)	92.29	92.63	41.00	112,632

Table 6: F1 scores of Ditto with the base BERT and ALBERT models on the 3 datasets where Ditto with DistilBERT does not outperform DeepMatcher+ (DM+), the SOTA matching models.

Datasets	DM+	Ditto (BERT)	delta	Ditto (ALBERT)	delta
DBLP-Google	94.70	94.80	(+0.10)	94.73	(+0.03)
Fodors-Zagats	100.00	100.00	0.00	100.00	0.00
Company	92.70	93.15	(+0.45)	92.89	(+0.19)

Training time. We plot the training time required by DeepMatcher and Ditto in Figure 6. We do not plot the time for Ditto(DA) because the DK optimization only pre-processes the data and adds

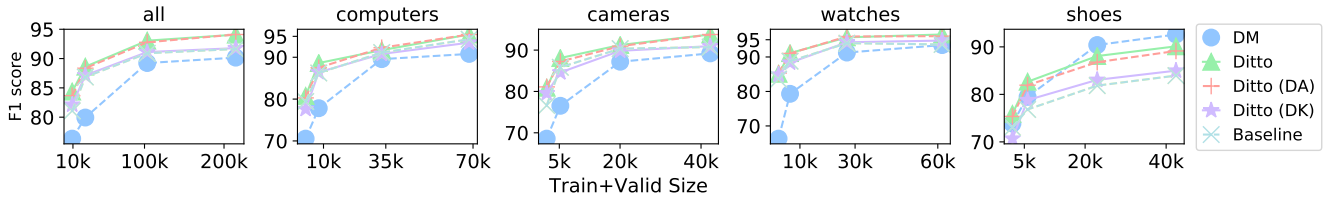

Figure 5: F1 scores on the WDC datasets of different versions of Ditto. **DM**: DeepMatcher.

Table 7: F1 scores on the WDC product matching datasets. The numbers for DeepMatcher (DM) are taken from [31].

Size	xLarge (1/1)		Large (1/2)		Medium (1/8)		Small (1/20)	
Methods	DM	Ditto	DM	Ditto	DM	Ditto	DM	Ditto
Computers	90.80	95.45	89.55	91.70	77.82	88.62	70.55	80.76
		+4.65		+2.15		+10.80		+10.21
Cameras	89.21	93.78	87.19	91.23	76.53	88.09	68.59	80.89
		+4.57		+4.04		+11.56		+12.30
Watches	93.45	96.53	91.28	95.69	79.31	91.12	66.32	85.12
		+3.08		+4.41		+11.81		+18.80
Shoes	92.61	90.11	90.39	88.07	79.48	82.66	73.86	75.89
		-2.50		-2.32		+3.18		+2.03
All	90.16	94.08	89.24	93.05	79.94	88.61	76.34	84.36
		+3.92		+3.81		+8.67		+8.02

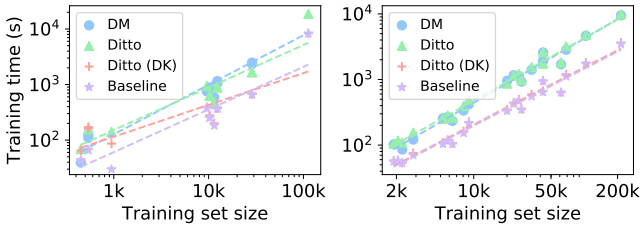

Figure 6: Training time vs. dataset size for the ER-Megallan datasets (left) and the WDC datasets (right). Each point corresponds to the training time needed for a dataset using different methods. Ditto(DK) does not use MixDA thus is faster than the full Ditto. DeepMatcher (DM) ran out of memory on the Company dataset so the data point is not reported. We omit Ditto(DA) in the figures because its running time is very close to Ditto's.

no more than 5% of training time. The running time ranges from 69 seconds (450 examples) to 5.2 hours (113k examples). Ditto has a similar training time to DeepMatcher although DistilBERT, which is used by Ditto, has a Transformer-based architecture that is deeper and more complex. The speed-up is due to DistilBERT and the fp16 optimization. Ditto with MixDA is about 2-3x slower than Ditto(DK) without MixDA. This is because MixDA requires additional time for generating the augmented pairs and computing with the LM twice. However, this overhead only affects offline training and does not affect online prediction.

4.4 Ablation study

Next, we analyze the effectiveness of each component (i.e., LM, SU, DK, and DA) by comparing Ditto with its variants without these optimizations. The results are shown in Table 5 and Figure 5.

The use of a pre-trained LM contributes to a large portion of the performance gain. In the ER-Magellan datasets (excluding Company), the average improvement of the baseline compared to DeepMatcher+ is 3.49, which accounts for 58% of the improvement of the full Ditto (6.0). While DeepMatcher+ and the baseline Ditto (essentially fine-tuning DistilBERT) are comparable on the Structured datasets, the baseline performs much better on all the Dirty datasets and the Abt-Buy dataset. This confirms our intuition that the language understanding capability is a key advantage of Ditto over existing EM solutions. The Company dataset is a special case because the length of the company articles (3,123 words on average) is much greater than the max sequence length of 256. The SU optimization increases the F1 score of this dataset from 41% to over 92%. In the WDC datasets, across the 20 settings, LM contributes to 3.41 F1 improvement on average, which explains 55.3% of improvement of the full Ditto (6.16).

The DK optimization is more effective on the ER-Magellan datasets. Compared to the baseline, the improvement of Ditto(DK) is 1.98 on average and is up to 9.67 on the Beer dataset while the improvement is only 0.22 on average on the WDC datasets. We inspected the span-typing output and found that only 66.2% of entry pairs have spans of the same type. This is caused by the current NER module not extracting product-related spans with the correct types. We expect DK to be more effective if we use an NER model trained on the product domain.

DA is effective on both datasets and more significantly on the WDC datasets. The average F1 score of the full Ditto improves upon Ditto(DK) (without DA) by 0.53 and 2.53 respectively in the two datasets. In the WDC datasets, we found that the span_del operator always performs the best while the best operators are diverse in the ER-Magellan datasets. We list the best operator for each dataset in Table 8. We note that there is a large space of tuning these operators (e.g., the MixDA interpolation parameter, maximal span length, etc.) and new operators to further improve the performance. Finding the best DA operators for EM is future work beyond the scope of this paper.

Table 8: Datasets that each DA operator achieves the best performance. The suffix (S)/(D) and (Both) denote the clean/dirty version of the dataset or both of them. All operators are applied with the entry_swap operator.

Operator	Datasets
span_shuffle	DBLP-ACM (Both), DBLP-Google (Both), Abt-Buy
span_del	Walmart-Amazon(D), Company, all of WDC
attr_del	Beer, iTunes-Amazon(S), Walmart-Amazon(S)
attr_shuffle	Fodors-Zagats, iTunes-Amazon(D)

5. CASE STUDY: EMPLOYER MATCHING

We present a case of applying Ditto to a real-world EM task. An online recruiting platform would like to join its internal employer records with newly collected public records to enable downstream aggregation tasks. Formally, given two tables A and B (internal and public) of employer records, the goal of the task is to find, for every record in table B, a record in table A that represents the same employer. Both tables have 6 attributes: name, addr, city, state, zipcode, and phone. Our goal is to find matching record pairs with both high precision and recall.

Basic blocking. Our first challenge is size of the datasets. As shown in Table 9, both tables are of nontrivial sizes even after deduplication. Thus, a naive pairwise comparison is not feasible. The first blocking method we designed is to only match companies with *the same zipcode*. However, since 60% of records in Table A do not have the zipcode attribute and some large employers have multiple sites, we use a second blocking method that returns for each record in Table B the top-20 most similar records in A ranked by the TF-IDF cosine similarity of name and addr attributes. We use

Table 9: Sizes of the two employer datasets to be matched.

	TableA		TableB		#Candidates
	original	deduplicated	original	deduplicated	Basic blocking
Size	789,409	788,094	412,418	62,511	10,652,249

the union of these two methods as our blocker, which produces 10 million candidate pairs.

Data labeling. We labeled 10,000 pairs sampled from the results of each blocking method (20,000 labels in total). We sampled pairs of high similarity with higher probability to increase the difficulty of the dataset to train more robust models. The positive rate of all the labeled pairs is 39%. We split the labeled pairs into training, validation, and test sets by the ratio of 3:1:1.

Applying Ditto. The user of Ditto does not need to extensively tune the hyperparameters but only needs to specify the domain knowledge and choose a data augmentation operator. We observe that the street number and the phone number are both useful signals for matching. Thus, we implemented a simple recognizer that tags the first number string in the addr attribute and the last 4 digits of the phone attribute. Since we would like the trained model to be robust against the large number of missing values, we choose the attr_del operator for data augmentation.

We plot the model's performance in Figure 7. Ditto achieves the highest F1 score of 96.53 when using all the training data. Ditto outperforms DeepMatcher (DM) in F1 and trains faster (even when using MixDA) than DeepMatcher across different training set sizes.

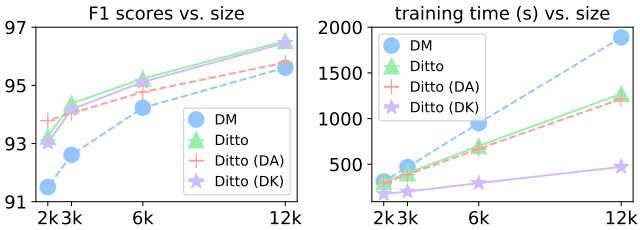

Figure 7: F1 scores and training time for the employer matching models.

Advanced blocking. Optionally, before applying the trained model to all the candidate pairs, we can use the labeled data to improve the basic blocking method. We leverage Sentence-BERT [34], a variant of the BERT model that trains sentence embeddings for sentence similarity search. The trained model generates a high-dimensional (e.g., 768 for BERT) vector for each record. Although this model has a relatively low F1 (only 92%) thus cannot replace Ditto, we can use it with vector similarity search to quickly find record pairs that are likely to match. We can greatly reduce the matching time by only testing those pairs of high cosine similarity. We list the running time for each module in Table 10. With this technique, the overall EM process is accelerated by 3.8x (1.69 hours vs. 6.49 hours with/without advanced blocking).

Table 10: Running time for blocking and matching with Ditto. Advanced blocking consists of two steps: computing the representation of each record with Sentence-BERT [34] (Encoding) and similarity search by blocked matrix multiplication [1] (Search). With advanced blocking, we only match each record with the top-10 most similar records according to the model.

	Basic Blocking	Encoding (GPU)	Search (CPU)	Matching (top-10)	Matching (ALL)
Time (s)	537.26	2,229.26	1,981.97	1,339.36	22,823.43

6. RELATED WORK

EM solutions have tackled the blocking problem [2, 6, 14, 28, 45] and the matching problem with rules [9, 13, 38, 44], crowdsourcing [16, 18, 43], or machine learning [37, 8, 3, 16, 20].

Recently, EM solutions used deep learning and achieved promising results [12, 15, 19, 27, 55]. DeepER [12] trains EM models based on the LSTM [17] neural network architecture with word embeddings such as word2vec [26] or GloVe [30]. DeepER also proposed a blocking technique to represent each entry by the LSTM's output. Our advanced blocking technique based on Sentence-BERT [34], described in Section 5, is inspired by this. Auto-EM [55] improves deep learning-based EM models by pre-training the EM model on an auxiliary task of entity type detection. Ditto also leverages transfer learning by fine-tuning pre-trained LMs, which are more powerful models in language understanding. We did not compare Ditto with Auto-EM in experiments because the entity types required by Auto-EM are not available in our benchmarks. However, we expect that pre-training Ditto with EM-specific data/tasks can improve the performance of Ditto further and is part of our future work. DeepMatcher introduced a design space for applying deep learning methods to EM. Following their template architecture, one can think of Ditto as replacing both the attribute embedding and similarity representation components in the architecture with a single pre-trained LM such as BERT, thus providing a much simpler overall architecture.

All systems, Auto-EM, DeepER, DeepMatcher, and Ditto formulate matching as a binary classification problem. The first three take a pair of data entries of the same arity as input and aligns the attributes before passing them to the system for matching. On the other hand, Ditto serializes both data entries as one input with structural tags intact. This way, data entries of different schemas can be uniformly ingested, including hierarchically formatted data such as those in JSON. Our serialization scheme is not only applicable to Ditto, but also to other systems such as Auto-EM, DeepMatcher, and DeepER. In fact, we serialized data entries to DeepMatcher under one attribute using our scheme and observed that DeepMatcher improved by as much as 1.94% on some datasets.

External knowledge is known to be effective in improving neural network models in NLP tasks [5, 40]. Instead of directly modifying the network architecture [46, 51] or the loss function [54] to incorporate domain knowledge, Ditto modularizes the way domain knowledge is incorporated by allowing users to specify and customize rules for preprocessing input entries. Data augmentation has been extensively studied in computer vision and has recently received more attention in NLP [24, 48, 50]. We designed a set of data augmentation operators suitable for EM and apply them with MixDA [24], a recently proposed DA strategy based on convex interpolation. To the best of our knowledge, this is the first time data augmentation has been applied to EM.

7. CONCLUSION

We present Ditto, the first EM system based on fine-tuned pre-trained Transformer-based language models. Ditto uses a simple architecture to leverage pre-trained LMs and is further optimized by injecting domain knowledge, text summarization, and data augmentation. Our results show that it outperforms existing EM solutions on all three benchmark datasets with significantly less training data. Ditto's good performance can be attributed to the improved language understanding capability mainly through pre-trained LMs, the more accurate text alignment guided by the injected knowledge, and the data invariance properties learned from the augmented data. We plan to further explore our design choices for injecting domain knowledge, text summarization, and data augmentation. In addition, we plan to extend Ditto to other data integration tasks beyond EM, such as entity type detection and schema matching with the ultimate goal of building a BERT-like model for tables.

8. REFERENCES

[1] F. Abuzaid, G. Sethi, P. Bailis, and M. Zaharia. To index or not to index: Optimizing exact maximum inner product search. In *Proc. ICDE '19*, pages 1250–1261. IEEE, 2019.

[2] L. R. Baxter, R. Baxter, P. Christen, et al. A comparison of fast blocking methods for record. 2003.

[3] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. KDD '03*, pages 39–48, 2003.

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[5] Q. Chen, X. Zhu, Z.-H. Ling, D. Inkpen, and S. Wei. Neural natural language inference models enhanced with external knowledge. In *Proc. ACL '18*, pages 2406–2417, 2018.

[6] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2011.

[7] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What does BERT look at? an analysis of BERT's attention. In *Proc. BlackBoxNLP '19*, pages 276–286, 2019.

[8] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proc. KDD '02*, pages 475–480, 2002.

[9] N. Dalvi, V. Rastogi, A. Dasgupta, A. Das Sarma, and T. Sarlos. Optimal hashing schemes for entity matching. In *Proc. WWW '13*, pages 295–306, 2013.

[10] S. Das, A. Doan, P. S. G. C., C. Gokhale, P. Konda, Y. Govind, and D. Paulsen. The magellan data repository. `https://sites.google.com/site/anhaidgroup/projects/data`.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL-HLT '19*, pages 4171–4186, 2019.

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

[13] A. Elmagarmid, I. F. Ilyas, M. Ouzzani, J.-A. Quiané-Ruiz, N. Tang, and S. Yin. NADEEF/ER: generic and interactive entity resolution. In *Proc. SIGMOD '14*, pages 1071–1074, 2014.

[14] J. Fisher, P. Christen, Q. Wang, and E. Rahm. A clustering-based framework to control block sizes for entity resolution. In *Proc. KDD '15*, pages 279–288, 2015.

[15] C. Fu, X. Han, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong. End-to-end multi-perspective matching for entity resolution. In *Proc. IJCAI '19*, pages 4961–4967. AAAI Press, 2019.

[16] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *Proc. SIGMOD '14*, pages 601–612, 2014.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] A. M. E. W. D. Karger and S. M. R. Miller. Human-powered sorts and joins. *PVLDB*, 5(1), 2011.

[19] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa. Low-resource deep entity resolution with transfer and active learning. In *Proc. ACL '19*, pages 5851–5861, 2019.

[20] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[21] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1-2):484–493, 2010.

[22] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proc. ICLR '20*, 2020.

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

[24] Z. Miao, Y. Li, X. Wang, and W.-C. Tan. Snippext: Semi-supervised opinion mining with augmented data. In *Proc. WWW '20*, 2020.

[25] R. Mihalcea and P. Tarau. TextRank: Bringing order into text. In *Proc. EMNLP '04*, pages 404–411, 2004.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS '13*, pages 3111–3119, 2013.

[27] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proc. SIGMOD '18*, pages 19–34, 2018.

[28] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas. Blocking and filtering techniques for entity resolution: A survey. *arXiv preprint arXiv:1905.06167*, 2019.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS '19*, pages 8024–8035, 2019.

[30] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proc. EMNLP '14*, pages 1532–1543, 2014.

[31] A. Primpeli, R. Peeters, and C. Bizer. The WDC training dataset and gold standard for large-scale product matching. In *Companion Proc. WWW '19*, pages 381–386, 2019.

[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[33] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[34] N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proc. EMNLP-IJCNLP '19*, pages 3982–3992, 2019.

[35] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proc. EMNLP '15*, 2015.

[36] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Proc. EMC^2 '19*, 2019.

[37] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. KDD '02*, pages 269–278, 2002.

[38] R. Singh, V. V. Meduri, A. Elmagarmid, S. Madden, P. Papotti, J.-A. Quiané-Ruiz, A. Solar-Lezama, and N. Tang.

Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202, 2017.

[39] Spacy. https://spacy.io/api/entityrecognizer.

[40] Y. Sun, S. Wang, Y. Li, S. Feng, X. Chen, H. Zhang, X. Tian, D. Zhu, H. Tian, and H. Wu. ERNIE: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*, 2019.

[41] I. Tenney, D. Das, and E. Pavlick. BERT rediscovers the classical NLP pipeline. In *Proc. ACL '19*, pages 4593–4601, 2019.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. NIPS '17*, pages 5998–6008, 2017.

[43] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[44] J. Wang, G. Li, J. X. Yu, and J. Feng. Entity matching: How similar is similar. *PVLDB*, 4(10):622–633, 2011.

[45] Q. Wang, M. Cui, and H. Liang. Semantic-aware blocking for entity resolution. *TKDE*, 28(1):166–180, 2015.

[46] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua. KGAT: Knowledge graph attention network for recommendation. In *Proc. KDD '19*, page 950âĂŞ958, 2019.

[47] WDC Product Data Corpus. http://webdatacommons.org/largescaleproductcorpus/v2.

[48] J. Wei and K. Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proc. EMNLP-IJCNLP '19*, pages 6382–6388, 2019.

[49] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. Huggingface's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[50] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

[51] B. Yang and T. Mitchell. Leveraging knowledge bases in LSTMs for improving machine reading. In *Proc. ACL '17*, pages 1436–1446, 2017.

[52] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. XLNet: Generalized autoregressive pretraining for language understanding. In *Proc. NeurIPS '19*, pages 5754–5764, 2019.

[53] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *Proc. ICLR '18*, 2018.

[54] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu. ERNIE: Enhanced language representation with informative entities. In *Proc. ACL '19*, pages 1441–1451, 2019.

[55] C. Zhao and Y. He. Auto-EM: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *Proc. WWW '19*, pages 2413–2424, 2019.