Neural Locality Sensitive Hashing for Blocking in Entity Blocking

Anonymous ACL submission

Abstract

Locality-sensitive hashing (LSH) is an algorithmic technique that hashes similar input items into the same "buckets" with high probability. It is a basic primitive in several large-scale data processing applications, including nearest-neighbor search, entity resolution, clustering, etc. In this work, we focus on the blocking phase in the entity resolution task. The goal of blocking is to avoid comparing all entity pairs by filtering out unmatched pairs. For this purpose, existing LSH functions that are based on generic similarity metric like Jaccard similarity, can only capture the occurrence of words while the semantics of the texts are ignored. On the other hand, several work have proposed to use language models to vectorize the data items and use the similarity of embeddings to find candidate pairs. However, it is still a challenge to fine-tune the language models so that the obtained embeddings can precisely capture the similarity of item pairs for ranking purpose. To this end, we propose NLSHBlock (Neural-LSH Block), a blocking approach that is based on pre-trained language models and fine-tuned with a novel LSH-inspired loss function. We evaluate the performance of Neural-LSH on the blocking stage of entity resolution, and show that it out-performs existing methods by a large margin on a wide range of datasets.

1 Introduction

Entity Resolution (ER) is a field of study dedicated to finding items that belong to the same entity, and is an essential problem in NLP and data mining (Rajaraman and Ullman, 2011; Getoor and Machanavajjhala, 2012; Konda et al., 2016)

For example, Grammarly's plagiarism checker detects plagiarism from billions of web pages and academic databases, Google News finds all versions of the same news from difference sources to have a comprehensive coverage, AWS has an Identity Resolution service for linking disparate customer identifiers into a single customer profile.

In such applications, an entity, either a customer profile or a piece of news, is essentially a text item consisting of words, and a pair of items is called a match if the pair represents the same real-world entity. A naive approach to finding matching items is to compare each pair of items. This approach however is computationally expensive when the size of the dataset is large due to the quadratic growth in computation time. In the literature, the pipeline of ER usually has two major components: blocking and matching. The blocking component finds candidate pairs where the two items are likely to be matches while discarding unmatched pairs, and the matching component determines if a candidate pair is really a match.

Locality-Sensitive Hashing (LSH) (Rajaraman and Ullman, 2011) can be applied in blocking to find candidate pairs with high Jaccard similarity by using MinHash functions. However, Jaccard similarity cannot effectively find candidate pairs in all use cases because it does not understand the latent semantics of the text. Many blocking techniques based on string and set similarity (Gokhale et al., 2014; Das et al., 2017; Simonini et al., 2019, 2016) also suffer from similar problems.

Most recently, deep learning models, especially the deep language models, have shown great success in entity resolution by achieving state-of-the-art performance in accuracy (Thirumuruganathan et al., 2021; Wang et al., 2022; Peeters and Bizer, 2022; Li et al., 2021; Miao et al., 2021). With the help of deep pre-trained language models, entities can be represented by embeddings to capture the semantics, and similar entities can be found by comparing the similarity of their embeddings. Nonetheless, it is still a challenge to fine-tune the language models specifically for blocking so that the obtained embeddings can precisely capture the similarity of item pairs for ranking purpose.

In this work, we present a novel approach called Neural Locality Sensitive Hashing Blocking

(NLSHBlock), which uses deep neural networks to approximate the locality preserving hashing functions. The main components of NLSHBlock include a language model for encoding data items and projection layers for projecting a high-dimensional vector to a scalar value. The scalar value is the hash value calculated by the approximated LSH function. NLSHBlock generates embeddings for data items, and finds candidate item pairs by LSH-based search technique on their embeddings. We design a loss function that fine-tunes the language model with the help of the projection layers, so that NLSHBlock can approximate any LSH function. After training, the language model is calibrated to map data items to a high-dimensional space where the similarity of these items is precisely preserved. Concisely, the objective of the fine-tuning is to maximize the probability that the scalar values of a pair of matched items are nearby, and also to maximize the probability that the hash values of an unmatched pair of items are far apart.

Existing deep learning models have explored different learning objectives for blocking. DL-Block (Thirumuruganathan et al., 2021) is a deep learning framework achieving state-of-the-art results on blocking based on a variety of deep learning techniques, including self-supervised learning. However, its self-supervision is either based on auxiliary tasks or the triplet loss on randomly generated training examples. Sudowoodo (Wang et al., 2022) is a multi-purpose Data Integration and Preparation framework based on self-supervised contrastive representation learning and pre-trained language models. It utilizes contrastive loss and data augmentation to learn representations for blocking. Peeters et al. (Peeters and Bizer, 2022) propose R-SupCon, a supervised contrastive learning model for product matching, and use the learned embeddings for blocking. However, the learning objective of R-SupCon is designed for matching, which is essentially a binary classification task. With this objective, the model is not optimized for the blocking, where the embeddings need to precisely capture the similarity of data items. What's more, in some real-world applications, task-specific similarity measurements for the data items are designed for specific use cases. The above methods cannot precisely preserve the similarity under specified measurements. Designing hash functions for such similarity measurements is extremely hard, and existing models are mostly designed for general cases.

NLSHBlock tackles the above issues by learning to approximate locality sensitive hashing functions for data items under the specific similarity measurement. The merits of NLSHBlock includes:

- It captures the semantics of data items better than traditional LSH with the help of generic pre-trained language models.
- Its novel learning objective helps fine-tune the pre-trained language models specifically for capturing the similarity of data items.
- On a wide range of real-world datasets for evaluating entity resolution, it out-performs state-of-the-art deep learning models and the traditional LSH-based approach.

2 Related Work

Locality Sensitive Hashing. The LSH was originally proposed by Indyk and Motwani (1998) for in-memory approximate high-dimensional nearest neighbor search in the Hamming space. Later it was adapted for external memory use by Gionis et al. (1999), and the space complexity is reduced by a "magic radius". Datar et al. (2004) proposed the locality-sensitive hash functions based on p-stable distribution and extended LSH to Euclidian distance. Shrivastava and Li (2014) developed asymmetric LSH for maximum inner product search. Andoni et al. (2015) designed an optimal LSH for Angular distance. Lv et al. (2007) proposed multi-probe LSH that checks both data objects falling in the same bucket as the query object, and data objects in buckets that have high success probability. C2LSH (Gan et al., 2012) is a different LSH schemed where the candidates are found by counting the number of collisions.

Recently, learned LSH has shown success on the nearest neighbor search of high-dimensional data. Neural LSH (Dong et al., 2020) uses neural networks to predict which bucket to hash for each input data item. Data-dependent hashing is another research direction where the random hash function is chosen after seeing the given datasets, and achieves lower time complexity (Andoni and Razenshteyn, 2015; Andoni and Razensteyn, 2016; Bai et al., 2014; Andoni et al., 2018). These work are dedicated to achieve tighter lower bound for time complexity of LSH methods.

Blocking in Entity Matching. Entity Matching (EM) is an essential research problem that has

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

Product Name	Manufacturer	Price
instant immersion spanish deluxe 2.0	topics entertainment	39.99
adventure workshop 4th-6th grade 7th edition	encore software	29.99
sharp printing calculator	sharp el1192b	45.63

match

unmatch

Product Name	Manufacturer	Price
encore inc adventure workshop 4th-6th grade 7th edition	encore	26.49
adventure workshop 4th-6th grade 8th edition	NULL	39.99
new-sharp shr-el1192bl two-color printing calculator 12-digit lcd black red	sharp	45.99

been extensively studied over past decades (Getoor and Machanavajjhala, 2012; Konda et al., 2016). The goal of EM is to find data items that represent the same real-world entity. Blocking and matching are two main steps in an EM pipeline, and many deep learning methods have been proposed for the matching step, including (Kasai et al., 2019; Peeters et al., 2020; Li et al., 2021; Miao et al., 2021; Akbarian Rastaghi et al., 2022; Yao et al., 2022). The blocking step is equally important, but so far very few deep learning methods are dedicated to it. The purpose of blocking step is to reduce the number of pairs for the matching step, where the potential number of pair comparisons could be as large as the square of the dataset size. For instance, if there are a million items in the dataset, a naive approach will compare half a trillion pairs. A simple pairwise comparison function averaging 10 micro-second would take more than 57 days to process all the pairs. What's more, in real-world applications the comparison functions may involve complex components such as deep neural networks (Wang et al., 2022; Li et al., 2021) for better accuracy, and they usually need to deal with millions, or even billions, of items. Therefore, using a naive approach is not computationally feasible. The goal of blocking is to find as many true matched pairs as possible while keeping the candidate set small. Example techniques include rule-based blocking (Gokhale et al., 2014; Das et al., 2017), schema-agnostic blocking (Simonini et al., 2019), meta-blocking (Simonini et al., 2016), deep learning approaches (Zhang et al., 2020; Thirumuruganathan et al., 2021), and LSH-based blocking technique that scale to billions of items for entity matching (Borthwick et al., 2020). Most recently, people resort to pre-trained language models to capture the semantics of text items. For example, BERT-based models are fine-tuned by contrastive learning methods and/or labeled data, and then generate embeddings for items. Then, similar item pairs can be found by performing similarity search on the embeddings (Li et al., 2021; Wang et al., 2022; Peeters and Bizer, 2022).

Entity blocking can also be considered an Information Retrieval (IR) task. Recent deep learning methods (Tonellotto, 2022) in the IR literature such as DPR (Karpukhin et al., 2020), GTR (Ni et al., 2021), and Contriever (Izacard et al., 2021) learn dense representation for documents, and candidate pairs can be found by performing similarity search on their dense representations using FAISS (Johnson et al., 2019).

3 Methodology

In this section, we lay out a formal problem definition, discuss the pipeline for solving the blocking task, and describe our proposed ranking loss inspired by locality sensitive hashing.

3.1 Blocking in Entity Resolution

A common scenario of Entity Resolution involves two tables A and B of data items, and the goal is to find all pairs (x, y) where $x \in A \wedge y \in B$ and both x and y refer to the same real-world entity. Such pairs are also called matches. We assume that the two tables have the same schema, i.e. the corresponding columns refer to the same type.

Figure 1 shows an example where two tables contain product items, and they both of them have the same schema ("Product Name", "Manufacturer", "Price") for their items. The solid arrow indicates that the second item in the left table matches the first item in the right table. The dashed arrow indicates that the second item in the left table does not match the second item in the right table.

Definition 3.1 (Blocking)**.** Given two collections A and B of data items, the blocking refers to the process of finding a candidate set of pairs $C = \{(x, y)|x \in A, y \in B\}$, where each pair is likely to be a match.

Let G be the ground-truth matches, an ideal blocking solution maximizes the recall $|C \cap G|/|G|$, and minimizes the size of candidate set size $|C|$. With a fixed recall, a smaller $|C|$ means less non-matching pairs are included and a higher precision.

Definition 3.2 (Embedding)**.** Given a collection A of data items, a d-dimensional embedding model M_{emb} takes every data item $x \in D$ as input and outputs a real vector $M_{\mathsf{emb}}(x) \in \mathbb{R}^d$. Given a similarity function sim, e.g., euclidean distance, for every pair of data items (x, x'), the value of sim(x, x') is large if and only if (x, x') matches.

For simplicity, we assume all output vectors are normalized, i.e. the L-2 norm $\|M_{\text{emb}}(x)\|_2 = 1$ for every data item $x \in D$.

3.2 Locality Sensitive Hashing

The high-level idea behind LSH is to hash items into buckets with some hash functions that are developed by domain experts to maximize the collision (being hashed into the same bucket) possibility among similar items and minimize the collision possibility of dissimilar items.

Now we present the definition of Locality Sensitive Hashing (LSH) (Rajaraman and Ullman, 2011; Zhao et al., 2014; Gionis et al., 1999). An LSH family $\mathcal{F}$ is defined for a metric space $\mathcal{M} = (M, d)$, a threshold $R > 0$, an approximation factor $c > 1$, and probabilities P_1 and P_2. In the metric space $\mathcal{M}$, M is the representation space of the data, and d is the distance function in this space. This family $\mathcal{F}$ is a set of functions $h\colon M \to S$ that map elements of the metric space to buckets $s \in S$. An LSH family must satisfy the following conditions for any two points $p, q \in M$ and any hash function h chosen uniformly at random from $\mathcal{F}$:

- if $d(p, q) \leq R$, then $h(p) = h(q)$ (i.e., p and q collide) with probability at least P_1,
- if $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability at most P_2.

When $P_1 > P_2$, such a family $\mathcal{F}$ is called (R, cR, P_1, P_2)-sensitive. If we consider a deep neural network as a hash function that maps data items to buckets, then we expect the collision probability of similar data items are high, and the collision probability of dissimilar data items are low. Instead of designing hash functions that satisfy the constraint, we propose to train a deep neural network to maximize P_1 and minimize P_2, and this process can be considered as neuralizing the LSH.

3.3 Neuralizing LSH

The core idea of neuralizing LSH is to train a deep neural network to approximate the locality preserving hash functions. Instead of using MinHash to approximate Jaccard Similarity, or other hash functions that are designed for approximating generic similarity metrics to decide which bucket to hash, we use deep neural networks to approximate the process. Our rationale is that the locality preserving hash functions are sophisticated and designed by experts, and it is extremely difficult to design such hash functions for ad-hoc distance functions that are used in many real-world applications. In Figure 3, we give an example of such ad-hoc distance functions, which is a rule-based similarity measurement for matching entities consisting of containment,[1] symmetric difference,[2] and Jaccard Similarity. It can adapt to specific use cases by configuring the weights of different similarity measurement and adding more components. Manually designing hashing techniques that preserve similarity for such metric rules is impractical.

The full pipeline of NLSHBlock is shown in Figure 4. Given two tables of data items, we first serialize the data items, then use the embedding model M_{emb} to encode the items. Next, we use a neural network with three projection layers to map embeddings to hash values. We denote this process as Nueralized Locality Sensitive Hashing ($NLSH$). Given a collection of data items X and a similarity metric M, the training of the M_{emb} involves the original data X_{ori}, augmented version X_{aug}, and dissimilar items Y_{neg}. The details will be discussed in a later subsection. An optional component is the contrastive learning as shown in the dashed box. E_{ori} and E_{aug} are embeddings of X_{ori} and X_{aug} respectively, and constrastive loss functions can be applied for fine-tuning M_{emb}.

3.4 Encode the data items

To use pre-trained language models for processing data items, the raw texts are first serialized the same way as in (Li et al., 2021; Miao et al., 2021; Wang et al., 2022): for each data entry $e = (attr_i, val_i)_{1 \leq i \leq k}$, we let serialize$(e)$ ::= [COL] $attr_1$ [VAL] val_1 ... [COL] $attr_k$ [VAL] val_k.

[COL] and [VAL] are special tokens that indicate the beginning of attribute names and values respectively. Figure 2 shows an example of serializing a conference paper with four attributes.

Next, the serialized texts are fed into an embedding model M_{emb} to get one embedding for each data item as shown in the Figure 4. In this work, we consider a pre-trained Transformer-based language model, such as BERT (Devlin et al., 2018) and its variants. Transformer-based language models generate embeddings that are highly contextualized, and capture better understanding of texts compared to traditional word embeddings.

[1] Intersection size divided by the size of the smaller set

[2] The symmetric difference is equivalent to the union of both relative complements

Figure 2: An example for serialization of data items

Authors	Title	Venue	Year
Kleissner, Charly	Enterprise Objects Framework: a Second Generation Object-relational Enabler	Proceedings of the ACM International Conference on Management of Data	1995

serialization

[COL] Authors [VAL] Charly Kleissner [COL] Title [VAL] Enterprise Objects Framework : a Second Generation Object-relational Enabler [COL] Venue [VAL] Proceedings of the ACM International Conference on Management of Data [COL] Year [VAL] 1995

Figure 3: An Example of Customized Similarity Metric

Product Name	Manufacturer	Price
instant immersion spanish deluxe 2.0	topics entertainment	39.99
adventure workshop 4th-6th grade 7th edition	encore software	29.99
sharp printing calculator	sharp el1192b	45.63

Customized Similarity

```
Similarity_Metric_Rule(item A, item B, attributes, weights):
    score = 0.0
    For attr in attributes:
        score += containment(A[attr], B[attr]) * weights[con]
        score += symmetric_difference(A[attr], B[attr]) * weights[sym]
        score += jaccard(A[attr], B[attr]) * weights[jac]
    return score
```

Existing works (Wang et al., 2022; Li et al., 2021) have shown that using the pre-trained language models without fine-tuning to obtain embeddings is not the optimal option. Efforts have been made to fine-tune the pre-trained language models for the matching phase of entity resolution problem. However, fine-tuning for the blocking phase is not well-studied to the best of our knowledge.

After getting the embeddings, we use a neural network to project the high-dimensional embeddings into scalar values. The neural network consists of three layers, where the first layer matches the dimension of embeddings, second layer is configurable, and the last layer has a single node.

3.5 Training NLSHBlock

To train the embedding model M_{emb} for NLSHBlock, we use a tuple of three data items as each training example. Let sim be a similarity function for a metric M. In each tuple (p, q, r), p and q are similar data items, and r is dissimilar to p and q. Thus, we have $\text{sim}(p, q) < \text{sim}(q, r)$. The goal of the training to achieve $|NLSH(p) - NLSH(q)| < |NLSH(p) - NLSH(r)|$, and we propose a loss function for this purpose:

$$\mathcal{L}_{\mathcal{LSH}} = \max(R, |NLSH(p) - NLSH(q)|) \\ - \min(cR, |NLSH(p) - NLSH(r)|)$$

If the absolute difference of hash values of two items is smaller than a pre-defined threshold R, we call it a collision. The first term $\max(R, |NLSH(p) - NLSH(q)|)$ corresponds to the first condition of an LSH family, and we want to maximize the probability of collisions of similar data items. The second term $-\lambda \min(cR, |NLSH(p) - NLSH(r)|)$ corresponds to the second condition of an LSH family, and we want to minimize the collision probability of two dissimilar items. Figure 5 shows an ideal distribution of hash values of data items, where matching items are close-by and the items belonging to different entities are far apart. We will discuss details on how to select training tuples in the evaluation section.

An optional step of our training is to use the self-supervised learning to fine-tune the pre-trained language model before the training of NLSHBlock. It is easy to integrate existing models to NLSHBlock. For example, Sudowoodo (Wang et al., 2022) uses self-supervised contrastive learning for fine-tuning the language model. It adapts Barlow Twins (Zbontar et al., 2021) and SimCLR (Chen et al., 2020) as its self-supervision loss, and uses Data Augmentation (DA) operators for generating distorted versions of the same item for robust representation learning. Examples of such DA operators include randomly removing a few words, swapping the positions of a few words, and token embedding level cutoff (Shen et al., 2020). Such operators are shown to not change the semantics of the data items in previous works, and thus can provide valid contrastion. This self-supervised learning can also be applied to NLSHBlock, and is an optional component.

3.6 Blocking

After M_{emb} is fine-tuned, we apply the embedding model M_{emb} on each data item and get the high-dimensional vector. We note that LSH is also commonly used for nearest neighbor search on high-dimensional data (Andoni et al., 2018; Gan et al., 2012). Then, we use a similarity search library such as FAISS that supports LSH (Johnson et al., 2019) to find the k most similar items for every input as the candidate set, where k is a configurable parameter.

4 Evaluations

We evaluate the performance of Neural-LSH on real-world datasets for blocking in entity resolu-

Figure 4: Architecture of Neural-LSH. The input tables are serialized to text sequences first. The training involves generating augmented sequences and randomly sampling negative examples. After trained with the loss fuction $\mathcal{L}_{\mathcal{LSH}}$, the model M_{emb} will generate embeddings for finding candidate pairs with LSH-based similarity search.

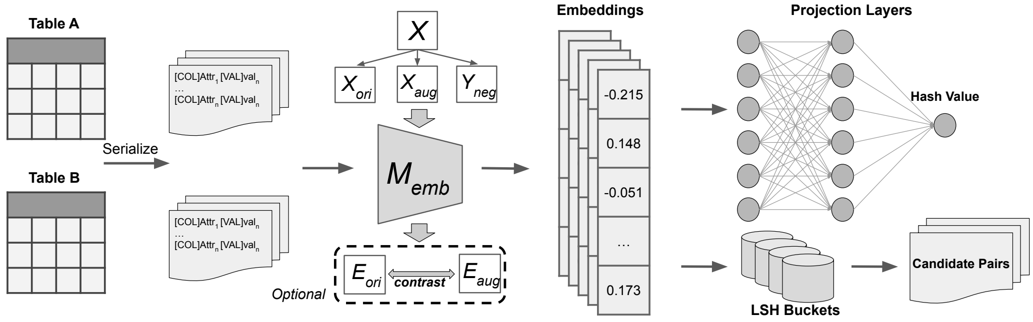


Figure 5: visualization of ideally hashed items

Table 1: Statistics of datasets.

Datasets	TableA	TableB	Groundtruth Pairs
Abt-Buy (AB)	1,081	1,092	1,028
Amazon-Google (AG)	1,363	3,226	1,167
DBLP-ACM (DA)	2,616	2,294	2,220
DBLP-Scholar (DS)	2,616	64,263	5,347
Walmart-Amazon (WA)	2,554	22,074	962

tion. The selected real-world datasets are widely used for evaluating the performance of entity in previous studies. They and are provided by (Mudgal et al., 2018) and publicly available (AnHai'sGroup). Their license allows private use.

4.1 Implementation Details

We implemented NLSHBlock using PyTorch (Paszke et al., 2019) and Huggingface (Wolf et al., 2020). The pre-trained language model we use is RoBERTa-base (Liu et al., 2019) and the optimizer is AdamW. The maximum input token length for RoBERTa-base is set to 128. The projector dimension is set to 4096 and batch size is 64. The learning rate is set to 10^{-5}, and we used linear learning rate scheduler. The projection layers of the NLSHBlock model is a 4096×4096×1 network, and weights are randomly initialized by default in torch, which follows a uniform distribution. The total number of parameters of our model is 125,236,993. The parameters in the loss function R and c are set as 0.01 and 3 respectively, and they are selected by grid search. We trained the model for 150 epochs and report the performance on the best epoch. The machine we used has a 12-core AMD Ryzen CPU, 32GB main memory, and 3 RTX 3090s (each with 24GB memory). For blocking, we construct the candidate pairs set by finding top similar items for each item and compare the performance with baselines by setting a target recall.

4.2 Datasets

The statistics of the datasets are shown in Table 1. These datasets include various domains such as products, publications, and businesses. In each dataset, there are two entity tables A and B, and blocking in entity resolution finds candidate record pairs across the two tables. The goal of blocking is to find as many true matching pairs as possible while minizing the number of candidate pairs. During the serialization, we use all the attributes and values for each data item.

We design similarity metric rules that are similar to the example in Figure 3 for the datasets. Suppose we have a collection of products from difference sources whose attributes include "name", "description", and "price". In some sources, the "name" only contains the product name, while other sources may include product details in the "name" attribute. Thus, the Jaccard similarity and symmetric difference should have lower weights and the containment score should have higher weight.

Each training example for NLSHBlock is a tuple (p, q, r), where p and q are similar items and r is a dissimilar one. There are two sources of similar item pairs: labeled data and data augmentation. All of the above public datasets contain labeled data, and we only used 60% of them for training. We generate augmented version of data items by a variety of operators, including randomly shuffling the words, randomly deleting a small portion of the words, and moving words across the attributes. For each similar item pair, we construct 10 tuples by select 10 dissimilar items. The dissimilar items are randomly selected and filtered by the metrics. The ratio between the number of training tuples and the total number of pairs in the dataset are 2.8%, 0.97%, 0.55%, 0.14%, 0.2% for AB, AG, DA, DS, and WA respectively.

Table 2: Comparison of Recall and the Number of Candidate Pairs

Dataset	AB			AG			DA			DS			WA		
	R	P	F1	R	P	F1	R	P	F1	R	P	F1	R	P	F1
HDB	84.0	1.5	2.94	97	0.1	0.2	99.6	29.5	45.5	97.7	1.6	3.15	94.7	0.32	0.64
DL-Block	88.0	4.2	8.0	97.1	1.66	3.27	99.6	16.9	28.9	98.1	1.34	2.64	92.2	1.74	3.4
Contriever	88.0	27.7	42.1	97.3	4.4	8.4	99.6	13.8	24.2	99.2	4.13	7.92	94.4	1.37	2.71
Sudowoodo	89	27.9	42.5	97.3	2.35	4.58	99.6	19.3	32.3	98.4	2.05	4.01	95	2.07	4.05
NLSHBlock-u	89.6	42.3	57.4	97.1	3.51	6.78	99.6	32.1	48.6	98.2	2.72	5.3	95.1	4.14	7.94
NLSHBlock	**94.4**	**88.9**	**91.6**	**97.3**	**8.8**	**16.1**	**99.6**	**48.2**	**65.0**	**98.9**	**4.11**	**7.90**	**96.3**	**4.20**	**8.04**
Δ	+5.4	+61	+49	+0.0	+4.4	+7.7	+0.0	+19	+20	-0.03	-0.02	-0.02	+1.3	+4.1	+4.0

Table 3: Comparison of the size of Candidate Sets

Datasets	AB	AG	DA	DS	WA
HDB	57,781	1,132,642	7,494	325,861	284,939
DL-Block	21,600	68,200	13,100	392,400	51,100
Contriever	3,276	25,808	16,058	128,526	66,222
Sudowoodo	3,276	48,390	11,470	257,052	44,148
NLSHBlock-u	2,184	32,260	6,882	192,789	22,074
NLSHBlock	1,092	12,904	4,588	128,526	22,074

4.3 Baselines

We include an LSH-based method HDB (Borthwick et al., 2020), state-of-the-art deep learning framework DL-Block (Thirumuruganathan et al., 2021), contrastive learning based method Sudowoodo (Wang et al., 2022), and the neural IR model Contriever (Izacard et al., 2021) as the baselines. We use NLSHBlock-u to denote the results of our method that only uses augmented training data, without labeled data.

4.4 Main Results on Blocking

We report Recall, Precision, F1 score, and the size of candidate set for each method on each dataset. Typically, a higher recall indicates that less true matching pairs are missing in the candidate set. A higher precision indicates that less unmatching pairs appear in the candidate set. F1 score combines Recall and Precision by their harmonic mean. In this work, we set a target recall and compare the precision and the size of candidate pairs. In general, if the recall is fixed, a smaller candidate set means the model excludes more unmatched pairs, which boosts the Precision and the F1 score.

Table 2 show the comparisons of different blocking methods on real-world datasets. We set the target recalls of the five datasets as 89%, 97%, 99%, 97%, and 94% resepectively for AB, AG, DA, DS, and WA. These target recalls are selected from DL-Block (Thirumuruganathan et al., 2021), which represent the top performance in its framework. For each measurement, a higher score means a better performance. In the baseline methods like DL-Block and Sudowoodo, for each item in table B, they find will have candidates from table A. For fair comparison, we follow the same strategy and use LSH for the similarity search. We use underline to highlight the best results of the baselines, use bold font to highlight the best results among all methods, and use colored numbers to show the performance differences of NLSHBlock against the best baseline on each dataset.

In a nutshell, NLSHBlock out-performs all baselines on all datasets except for DS, where NLSHBlock under-performs Contriever by a tiny margin. NLSHBlock out-performs NLSHBlock-u because labeled data helps.

On Abt-Buy, HDB does not perform well because it only captures Jaccard similarity of data items, and many true matching data items have very different text lengths. To achieve a high recall on this type of data, HDB has to include more candidate pairs, and thus its precision is negatively impacted. DL-Block performs better than HDB, because it is a deep learning method and captures more similarity between data items beyond Jaccard similarity. Sudowoodo and Contriever out-performs DL-Block by a large margin because they incorporate contrastive learning and learn more robust representations. NLSHBlock achieves the highest score in Recall, Precision and F1, because our novel learning objective enables NLSHBlock to precisely map items in a space where the similarity is well preserved.

On Amazon-Google, HDB does not perform well for the same reason. DL-Block is one order of magnitude better than HDB. NLSHBlock out-performs all baselines in terms of F1 score and reduces candidate set size at least by half, which is a significant improvement.

On DBLP-ACM, the data items are academic papers, where the true matching pairs have very high Jaccard similarity. To explain, if two academic paper from different datasets refer to the same work, they typically have very similar title, author list, venue, etc. Thus, the Jaccard similarity is very high for matching pairs. This dataset is relatively easy to solve and all of the methods can achieve higher than 99% recall. The traditional Jaccard similarity based method HDB performs better than DL-Block, Contriever, and Sudowoodo, because the later three methods added random noises during training. NLSHBlock out-performs all other methods because its loss function help better distinguish and rank similar items.

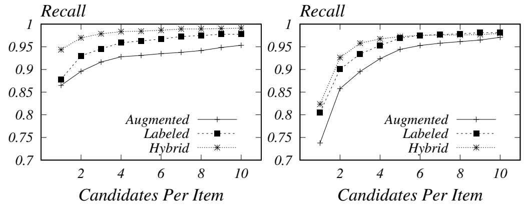
On DBLP-Scholar, the data items are also academic papers, and the LSH method HDB performs better than DL-Block, but a little worse than Sudowoodo. The performance of Contriever is slightly better than NLSHBlock because its negative sampling techniques let the model sees more diverse and larger number of negative examples during training, while NLSHBlock basically uses random negative sampling. Since this dataset is much larger than all others, seeing more negative examples helps Contriever gain some advantage.

On Walmart-Amazon, the performance differences among NLSHBlock and baselines are similar to Abt-Buy. The precision values in this dataset is about an order of magnitude lower than Abt-Buy. This is because the ratio of true matching pairs in WA is about 50× smaller than AB.

Table 3 lists the candidate sizes of different methods on all datasets. Among all methods, NLSHBlock requires much less candidate pairs to achieve the target recalls on all but one datasets. This is very important in practice, because the computation cost of the dominating pair-wise matching is significantly reduced.

In summary, NLSHBlock achieves up to 2.2× better F1 score compared to existing best methods, and consistently outperforms state-of-the-art methods on all datasets except on DS, where NLSHBlock slightly lags Contriever but still outperforms other baselines by a large margin. Given a target recall, NLSHBlock can reduce the number of candidate pairs by up to 67% compared to state-of-the-art methods, and thus significantly saves computation cost of the matching phase.

Figure 6: Comparison on Different Training Data



(a) AB (b) AG

4.5 Comparisons on Training Data

We compare the effect of using different training data for NLSHBlock in Figure 6. The three settings are: augmented data only, labeled data only, and hybrid data (using both augmented and labeled data). We selected two datasets Abt-Buy (AB) and Amazon-Google (AG) and report the relation between the size of candidate set and the recall under three settings. On both datasets, using only augmented data gives the worst performance, and using both types of data gives the best performance. Note that under all of these settings, NLSHBlock out-performs existing methods.

4.6 Limitations and Risks of NLSHBlock

Unlike traditional LSH methods, NLSHBlock cannot provide theoretical guarantee on the approximation ratio. Although it has empirically shown success on a wide range of real-world datasets, it might require additional adaptations on other use cases. To explain, NLSHBlock is able to capture the similarity of data items under a specific rule. However, the rules are designed by practitioners, and the augmentation operators might need further development to satisfy the specific rules. If a rule is not carefully designed and is ambiguous, NLSHBlock might not be able to perform well. Despite that, the practice of designing rules and adapting augmentation is far more feasible than designing sophisticate techniques similar to MinHash for Jaccard Similarity. Another limitation is that any learning-based model for entity blocking will require some training dataset, while alternative methods such as traditional LSH-based methods can be used without training.

5 Conclusion

In this paper, we propose NLSHBlock to approximate locality sensitive hashing functions for finding candidate pairs in entity resolution. NLSHBlock is able to preserve the distance under specified similarity metric rules, and is practical in real-world use cases. NLSHBlock out-performs existings methods for the blocking step of the entity resolution task on a wide range of real-world datasets.

References

Mehdi Akbarian Rastaghi, Ehsan Kamalloo, and Davood Rafiei. 2022. Probing the robustness of pre-trained language models for entity matching. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3786–3790.

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal lsh for angular distance. *Advances in neural information processing systems*, 28.

Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya Razenshteyn, and Erik Waingarten. 2018. Data-dependent hashing via nonlinear spectral gaps. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*, pages 787–800.

Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801.

Alexandr Andoni and Ilya Razensteyn. 2016. Tight lower bounds for data-dependent locality-sensitive hashing. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51, page 9. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

AnHai'sGroup. Public entity matching datasets. https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md. Accessed: 2022-12-15.

Xiao Bai, Haichuan Yang, Jun Zhou, Peng Ren, and Jian Cheng. 2014. Data-dependent hashing based on p-stable distribution. *IEEE Transactions on Image Processing*, 23(12):5033–5046.

Andrew Borthwick, Stephen Ash, Bin Pang, Shehzad Qureshi, and Timothy Jones. 2020. Scalable blocking for very large databases. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 303–319. Springer.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, pages 1431–1446.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yihe Dong, Piotr Indyk, Ilya P Razenshteyn, and Tal Wagner. 2020. Learning space partitions for nearest neighbor search. *ICLR*.

Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, pages 541–552.

Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: Theory, practice & open challenges. *PVLDB*, 5(12):2018–2019.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529.

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 *SIGMOD*, pages 601–612.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. In *ACL*, pages 5851–5861.

Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, and et al. 2016. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208.

Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2021. Deep entity matching with pre-trained language models. *PVLDB*, 14(1):50–60.

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. *arXiv preprint arXiv:1907.11692*.

Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. Citeseer.

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 *SIGMOD*, pages 1303–1316.

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 *SIGMOD*, pages 19–34.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y Zhao, Yi Luan, Keith B Hall, Ming-Wei Chang, et al. 2021. Large dual encoders are generalizable retrievers. *arXiv preprint arXiv:2112.07899*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Ralph Peeters and Christian Bizer. 2022. Supervised contrastive learning for product matching. *arXiv preprint arXiv:2202.02098*.

Ralph Peeters, Christian Bizer, and Goran Glavas. 2020. Intermediate training of BERT for product matching. In *DI2KG@VLDB*.

Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets*. Cambridge University Press.

Dinghan Shen, Mingzhi Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. 2020. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *arXiv preprint arXiv:2009.13818*.

Anshumali Shrivastava and Ping Li. 2014. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in neural information processing systems*, 27.

Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. 2016. BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *PVLDB*, 9(12):1173–1184.

Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. 2019. Schema-agnostic progressive entity resolution. *IEEE Trans. Knowl. Data Eng.*, 31(6):1208–1221.

Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen Glenn Fung, and AnHai Doan. 2021. Blocking in entity matching: A design space exploration. *PVLDB*, 14(11):2459–2472.

Nicola Tonellotto. 2022. Lecture notes on neural information retrieval. *arXiv preprint arXiv:2207.13443*.

Runhui Wang, Yuliang Li, and Jin Wang. 2022. Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation. *arXiv preprint arXiv:2207.04122*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

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*, pages 429–442.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR.

Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. 2020. Autoblock: A hands-off blocking framework for entity matching. In *WSDM*, pages 744–752.

Kang Zhao, Hongtao Lu, and Jincheng Mei. 2014. Locality preserving hashing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.