Neural Locality Sensitive Hashing for Entity Blocking

Runhui Wang* Luyang Kong† Yefan Tao† Andrew Borthwick† Davor Golac†
Henrik Johnson † Shadie Hijazi † Dong Deng* Yongfeng Zhang*

Abstract

Locality-sensitive hashing (LSH) is a fundamental algorithmic technique widely employed in large-scale data processing applications, such as nearest-neighbor search, entity resolution, and clustering. However, its applicability in some real-world scenarios is limited due to the need for careful design of hashing functions that align with specific metrics. Existing LSH-based Entity Blocking solutions primarily rely on generic similarity metrics such as Jaccard similarity, whereas practical use cases often demand complex and customized similarity rules surpassing the capabilities of generic similarity metrics. Consequently, designing LSH functions for these customized similarity rules presents considerable challenges. In this research, we propose a neuralization approach to enhance locality-sensitive hashing by training deep neural networks to serve as hashing functions for complex metrics. We assess the effectiveness of this approach within the context of the entity resolution problem, which frequently involves the use of task-specific metrics in real-world applications. Specifically, we introduce NLSHBlock (Neural-LSH Block), a novel blocking methodology that leverages pre-trained language models, fine-tuned with a novel LSH-based loss function. Through extensive evaluations conducted on a diverse range of real-world datasets, we demonstrate the superiority of NLSHBlock over existing methods, exhibiting significant performance improvements. Furthermore, we showcase the efficacy of NLSHBlock in enhancing the performance of the entity matching phase, particularly within the semi-supervised setting.

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 [40, 17, 26]. 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, and Amazon Web Service (AWS) has an Identity Resolution service for linking disparate customer identifiers from different sources into a single profile.

In such applications, an entity, whether it be 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 entity resolution usually has two major components: blocking and matching [35, 32, 45, 27]. The blocking component finds candidate pairs where the two items are likely to be matches, and the matching component determines if a candidate pair is really a match.

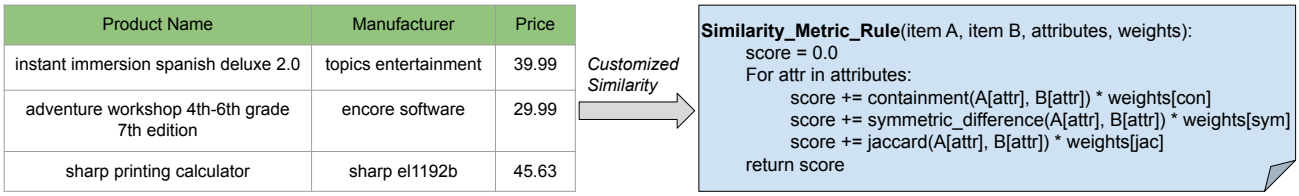
Locality-Sensitive Hashing (LSH) [40] can be applied in blocking to find candidate pairs with high Jaccard similarity by using MinHash functions. However, Jaccard similarity cannot effectively find candidate pairs in all use cases because it cannot effectively capture the latent semantics of the text. Many blocking techniques based on string and set similarity [19, 10, 44, 43] 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 [45, 47, 38, 27, 31]. With 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 the embeddings. For example, DL-Block [45] is a deep learning framework for blocking based on self-supervised learning, Sudowoodo [47] is a multi-purpose data integration and preparation framework based on contrastive representation learning and pre-trained language models, and R-SupCon [38] is a supervised contrastive learning model for product matching which uses the learned embeddings for blocking.

Nonetheless, in real-world applications, task-specific similarity measurements for the data items are often designed for specific use cases. Figure 1 shows an example of such ad-hoc distance functions, which is a rule-based similarity measurement for matching entities

*Rutgers University
†Amazon.com Services, Inc.

Figure 1: An Example of Customized Similarity Metric



consisting of containment,[1] symmetric difference,[2] and Jaccard Similarity. The above blocking methods cannot well preserve the similarity under specified measurements because: (1) designing hash functions for such similarity measurements is extremely hard, while existing models are mostly designed for general cases, (2) it is still a challenge to fine-tune language models specifically for entity blocking so that the obtained embeddings can capture the similarity of item pairs for blocking purpose.

In this work, we present a novel approach Neural Locality Sensitive Hashing for Blocking (NLSHBlock), which neuralizes locality preserving hashing functions based on deep pre-trained language models. NLSHBlock generates embeddings for input items, and finds candidate item pairs by k-Nearest-Neighbor search techniques on their embeddings. We design a loss function that fine-tunes the language model with the help of a projection layer, 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 preserved. Concisely, the objective of the fine-tuning is to maximize the probability that a pair of matched items are nearby in the high-dimensional space, meanwhile also to maximize the probability that any unmatched pair of items are far enough.

NLSHBlock tackles the aforementioned issues by learning to approximate locality sensitive hashing functions for data items under the specific similarity measurement. We note that NLSHBlock can also improve the performance of state-of-the-art ER methods such as Sudowoodo [47] on the matching task on the same test sets (i.e. sets of candidate pairs) of various real-world datasets, by facilitating pseudo labeling.

In short, the merits of NLSHBlock include:

- Its novel learning objective helps to fine-tune the pre-trained language models specifically for capturing the similarity of input items under task-specific metrics.
- 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.
- By providing better embeddings for pseudo-labeling, it can further boost the performance of entity matching of state-of-the-art methods.

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

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

2 Related Work

Locality Sensitive Hashing. LSH was originally proposed in [20] for in-memory approximate high-dimensional nearest neighbor search in the Hamming space. Later, it was adapted for external memory use by [18], and the space complexity is reduced by a "magic radius". Researchers also extended LSH to various distance metrics and improved its performance [12, 42, 2, 29, 16].

Recently, learned LSH has shown success on the nearest neighbor search of high-dimensional data. Neural LSH [13] 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 [4, 5, 6, 3]. These works are dedicated to achieve tighter lower bound for time complexity of LSH methods.

Blocking in Entity Resolution. Entity Resolution (ER) is an essential research problem that has been extensively studied over past decades [17, 26]. The goal of ER is to find data items that represent the same entity. Blocking and matching are two main steps in an ER pipeline, and many deep learning methods have been proposed for the matching step, including [24, 39, 27, 31, 1, 49]. The blocking step is equally important, and its goal is to include as many true matched pairs in a candidate set as possible (i.e. high recall) while keeping the candidate set small. Example techniques include rule-based blocking [19, 10], schema-agnostic blocking [44], meta-blocking [43], deep learning approaches [51, 45], and LSH-based blocking technique that scale to billions of items for entity matching [7]. 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 [27, 47, 38].

Entity blocking can also be considered from an

Figure 2: Entity Resolution: determine the matching entries from two datasets.

Table A

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
Match

Table B

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

Information Retrieval (IR) perspective. Recent deep learning methods [46] in the IR literature such as DPR [23], GTR [33], and Contriever [21] learn dense representation for documents, and candidate pairs can be found by performing similarity search on their dense representations using FAISS [22]. ColBERT [25, 41] achieves efficient and effective passage search via contextualized late interaction over BERT.

The matching process involves pairwise comparison aimed at identifying matched entity entries. Presently, deep learning-based techniques have shown great potential in this area, including DeepER [14], DeepMatcher [32], active learning based ER [24], Seq2SeqMatcher [34], HierMatcher [15], and pre-trained language model based methods (R-SupCon, Ditto, Rotom, Sudowoodo) [8, 39, 27, 31, 47]. In contrast to these recent methods, which optimize individual components separately, Sudowoodo [47] demonstrates promising results in both blocking and matching stages.

Our method differs from existing methods in that it captures the semantics of texts while our novel loss function aligns it with the desired similarity metrics better than other methods.

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 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 2 shows an example where two tables contain product items, and they both have the same schema ("Product Name," "Manufacturer," "Price") for their items. The solid arrows indicate matches between two tables, and the dashed arrow indicates a non-match.

DEFINITION 3.1. *[Blocking] Given two collections A and B of 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 items, a d-dimensional embedding model LM takes every item $x \in D$ as input and outputs a real vector $LM(x) \in R^d$. Given a similarity function* sim*, e.g., euclidean distance, for every pair of items (x, x'), the value of* $\mathsf{sim}(x, x')$ *is large if and only if (x, x') matches.*

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

3.2 Locality Sensitive Hashing The key 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) [40, 52, 18]. 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 $h: M \to S$ 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.

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

Figure 3: An example for serialization of 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

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. The example in Figure 1 can adapt to specific use cases by adding/removing components and configuring the weights of different similarity measurement. Suppose we have a collection of products from difference sources whose attributes include "name," "description," and "price". In some data sources, the "name" only contains the product name, while other sources may include product details in the "name" attribute. For this use case, the Jaccard similarity and symmetric difference should have lower weights and the containment score should have higher weight.

Figure 4 shows the NLSHBlock pipeline. Given two tables of items, we first serialize the items, and then use the embedding model LM 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 Neuralized Locality Sensitive Hashing ($NLSH$). Given a collection of items X and a similarity metric M, the training of the LM involves the original data X_{ori}, augmented version X_{aug}, and dissimilar items Y_{neg}. The details will be discussed in later subsections. An optional component is 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 LM.

3.4 Encode the items To use pre-trained language models for processing items, the raw texts are first serialized the same way as in [27, 31, 47]: for each data entry $e = (attr_i, val_i)_{1 \le i \le k}$, 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 3 shows an example of serializing a conference paper with four attributes.

Next, the serialized texts are fed into an embedding model LM to get one embedding for each item as shown in the Figure 4. In this work, we consider a pre-trained Transformer-based language model, specifically, the RoBERTa [28] model, which is a state-of-the-art BERT-based language model. Transformer-based language models generate embeddings that are highly contextualized, and capture better understanding of texts compared to traditional word embeddings [27]. Moreover, we fine-tune the language model component in our NLSHBlock, because recent research has shown that using the pre-trained language models without fine-tuning to obtain embeddings is not the optimal option [47, 27].

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.

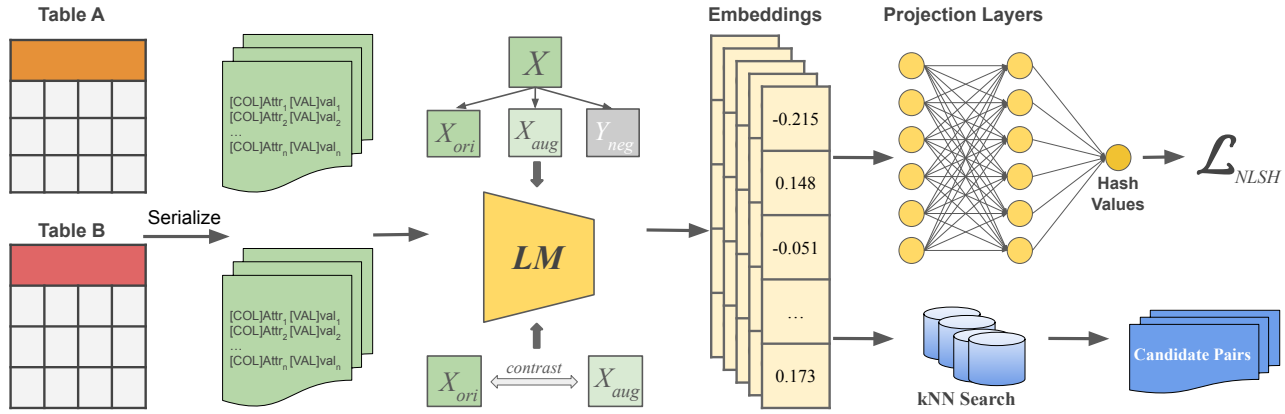
3.5 Training NLSHBlock To train the embedding model LM for NLSHBlock, we use a tuple of three items as each training example. Let sim be a similarity function for a metric M. In each tuple (p, q, r), p and q are similar items, and r is dissimilar to p and q. Thus, we have $\mathsf{sim}(p, q) > \mathsf{sim}(q,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

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

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 collision of similar 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 the hash values of items, where each entity is represented by a unique color. The matching items are close-by and share identical colors, and the items belonging to different entities are separated and colored differently.

In real-world applications, determining if a pair of items belong to the same entity depends on either an explicit similarity metric (e.g. the example metric in Figure 1) or an expert's knowledge. The latter can also be viewed as some more sophisticated similarity metric.

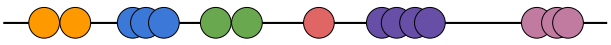
Figure 4: Architecture of Neural-LSH. The input tables are serialized to text sequences first. The training involves generating augmented sequences and randomly sampling negative examples. After training with the loss fuction $\mathcal{L}_{\mathcal{LSH}}$, the model LM will generate embeddings for finding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



To align the embedding model for capturing the desired similarity, the training tuples should be representative of such metrics.

The training examples for NLSHBlock is a collection of tuples. To construct each tuple, for an item p, we need to get a similar item q and a dissimilar item r. For similar item pairs, there are two sources: positively labeled pairs and Data Augmentation (DA). For DA, we follow the common practice and generate distorted version of items by a variety of operators that have been studied in previous work, including randomly shuffling the words, randomly deleting a small portion of the words, and moving words across the attributes [27, 31, 47]. For dissimilar item pairs, there are also two sources: negatively labeled pairs and random negative sampling. With a combination of DA for q and random negative sampling for r, NLSHBlock is trained in a self-supervised manner. When labeled pairs are needed for constructing training tuples, NLSHBlock is trained in a supervised manner. We will show experimental results for both self-supervised and supervised versions of our NLSHBlock approach.

Figure 5: Visualization of ideally hashed items



NLSHBlock uses contrastive learning objectives as a regularization technique. As demonstrated in [47], self-supervised contrastive learning can achieve state-of-the-art blocking performance in entity resolution. More specifically, we employ the widely used Barlow Twins [50] and SimCLR [9] as the loss function for contrastive learning in our approach.

3.6 Blocking After LM is fine-tuned, we apply the embedding model LM on each item and get the high-dimensional vector. Then, we use a similarity search library such as FAISS [22] to find the k most similar items for every input as the candidate set, where k is a configurable parameter. We note that the expected time complexity of graph-based approximate similarity methods is $O(n \log n)$ where n is the dataset size [30].

3.7 Pseudo Labeling for Entity Matching Though not the key focus of this paper, we would like to note that our NLSHBlock approach can not only improve blocking performance, but also can improve matching performance. The reason is because our method can be used to generate high-quality pseudo labels for training any matching model. For example, Sudowoodo is a state-of-the-art entity matching model with good performance on a wide range of datasets in a semi-supervised setting, and one key optimization technique is pseudo labeling [47], where a small amount of labeled pairs and the trained embedding model are used for automatically generating probabilistic labels and augmenting the small labeled set. NLSHBlock can boost the quality of the probabilistic labels because its embedding model is calibrated by the NLSH loss for better capturing the similarity of items, and thus generates better similarity-based thresholds for creating probabilistic labels. In the experiments, we will show the improved matching performance by leveraging the pseudo labels generated by our NLSHBlock method.

4 Evaluations

We evaluate the performance of Neural-LSH on real-world datasets for blocking in entity resolution. The selected real-world datasets are widely used for evaluating the performance of entity in previous studies. They are provided by [32] and publicly available [11].

4.1 Implementation Details We implemented NLSHBlock using PyTorch [36] and Huggingface Transformers [48]. The pre-trained language model we use is RoBERTa-base [28] and the optimizer is AdamW.

Table 1: Statistics of datasets.

Datasets	TableA	TableB	Matched
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

The maximum input token length for RoBERTa-base is set to 128. The projector dimension is set to 768 and batch size is 64. The learning rate is set to 10^{-5}, and we used linear learning rate scheduler with warm up. The projection layers of the NLSHBlock model is a 768×768×1 network, and weights are randomly initialized by default in pytorch, which follows a uniform distribution. The total number of parameters of our model is 125 million. The parameters R and c in the loss function NLSHLoss 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 has a 12-core AMD Ryzen CPU, 32GB memory, and RTX 3090 GPU (24GB). 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 and Training Examples 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 TableA and TableB, and blocking in entity resolution finds candidate record pairs across the two tables. All of the datasets contain human-labeled similar and dissimilar pairs, and thus the underlying similarity metric is an implicit and complex one hidden under the collective intelligence of the human annotators.

For training tuples, there are two sources of similar items: labeled data and data augmentation. All of the above public datasets contain labeled data, and we followed the standard train-validation-test ratio of 3:1:1 and use only labeled pairs in the trainset. Dissimilar items are randomly sampled. The total number of training tuples are 33k, 35k, 33k, 230k, and 113k for AB, AG, DA, DS, and WA respectively. We note that these numbers are far less than the total number of pairs (i.e. $|TableA|\times|TableB|$) in the corresponding datasets, and as a result, blocking on these datasets is trivial.

4.3 Baselines We consider two categories of baselines for comparison with NLSHBlock: methods that are specifically proposed for entity blocking, and methods that are proposed for information retrieval, which can also be applied for entity blocking.

HDB [7] is an LSH-based method for scalable blocking in entity resolution. It is applied in real-world cloud services for large scale datasets and uses Jaccard similarity as the metric.

DL-Block is a deep learning framework for entity blocking [45], which leverages a variety of deep learning techniques, including self-supervised learning and Transformers.

Sparkly [37] is a TF-IDF based method for entity blocking and achieves state-of-the-art results.

Sudowoodo [47] is a multi-purpose data integration and preparation framework based on contrastive representation learning, which is finetuned on RoBERTa-base [28].

Contriever [21] is a neural retrieval model that uses contrastive learning and Transformers to learn representations for documents.

ColBERT [25, 41] is a fast and accurate retrieval model, enabling scalable BERT-based search over large text collections. Its search is based on the similarity of token-level embeddings of the documents and achieves state-of-the-art performance on several question answering benchmark datasets.

Regarding the training of Contriver, for each dataset, we fine-tuned the checkpoint "facebook / contriever" with the all items. We followed the example script in the official Contriever repository on GitHub[3] and trained the model until the loss converged and became sufficiently small. For training ColBERT on our ER datasets, we followed the authors' instructions on their GitHub repository[4]: we set the query length to 128 to match NLSHBlock, and disabled the compression for best accuracy.

4.4 Main Results on Blocking We report Recall (R), Precision (P), F1 score, and the size of candidate set for each method on each dataset in Table 2 and Table 3. 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 accuracy and size of candidate pairs. 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 [45], which represent the best performance in its framework for each dataset. For each measurement, a higher score indicates a better performance. In the baseline methods like DL-Block and Sudowoodo, to obtain candidate pairs, they find candidates from TableA

[3]https://github.com/facebookresearch/contriever
[4]https://github.com/stanford-futuredata/ColBERT

Table 2: Comparison of Recall, Precision and F1 score of different methods. We use bold font to highlight the best method in each dataset and underline to highlight the second best method excluding NLSHBlock-s (NLSHBlock-s is the self-supervised version of NLSHBlock). In the last line, green numbers indicate better performance than the best baselines, and red numbers indicate an inferior performance compared to the best baselines.

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	2.9	97.0	0.1	0.2	99.6	29.5	45.5	97.7	1.6	3.2	94.7	0.3	0.6
DL-Block	88.0	4.2	8.0	97.1	1.7	3.3	99.6	16.9	28.9	98.1	1.3	2.6	92.2	1.7	3.4
Contriever	88.0	27.7	42.1	97.3	4.4	8.4	99.6	13.8	24.2	**99.2**	**4.1**	**7.9**	94.4	1.4	2.7
ColBERT	88.1	9.2	16.7	97.4	5.7	10.8	**99.7**	**48.2**	**65.0**	52.8	0.1	0.2	73.6	0.1	0.1
Sudowoodo	89.0	27.9	42.5	97.3	2.4	4.6	99.6	19.3	32.3	98.4	2.1	4.0	95.0	2.1	4.1
Sparkly	93.4	47.1	62.6	97.2	7.2	13.5	99.6	32.3	48.8	98.5	1.7	3.3	95.0	2.1	4.1
NLSHBlock-s	89.6	42.3	57.4	97.1	3.5	6.8	99.6	32.1	48.6	98.2	2.7	5.3	95.5	2.2	4.3
NLSHBlock	**94.4**	**88.9**	**91.6**	**97.8**	**8.8**	**16.2**	99.6	**48.2**	**65.0**	99.0	**4.1**	**7.9**	**96.3**	**4.2**	**8.0**
Δ	+1.0	+42	+29	+0.4	+1.6	+2.7	-0.1	+0.0	+0.0	-0.2	+0.0	+0.0	+1.3	+2.1	+3.9

Table 3: Comparison of the size of candidate sets. We use bold font to highlight the best method in each dataset and underline to highlight the second best method (excluding NLSHBlock-s). K=1,000, M=100K

Datasets	AB	AG	DA	DS	WA
HDB	57,781	1.1M	7,494	326K	285K
DL-Block	21,600	68,200	13,100	392K	51K
Contriever	3,276	25,808	16,058	**129K**	66K
ColBERT	9,828	19,956	**4,588**	3.2M	1.1M
Sudowoodo	3,276	48,390	11,470	257K	44K
Sparkly	2,184	16,130	6,877	321k	44K
NLSHBlock-s	2,184	32,260	6,882	193K	22K
NLSHBlock	**1,092**	**12,904**	**4,588**	**129K**	**22K**

for each item in TableB. For fair comparison, we follow the same strategy.

Table 2 show the comparisons of different blocking methods on real-world datasets. We use bold font to highlight the best results among all methods and use underline to highlight the second best results. The colored numbers are used to show the performance differences of NLSHBlock (supervised training) against the best competitor in each dataset. The performance of NLSHBlock-s (self-supervised training) is also shown.

In a nutshell, NLSHBlock out-performs all baselines by a large margin in terms of F1 score on a majority of datasets. On DA and DS, NLSHBlock is the runner-up and only slightly under-performs the best competitor in terms recall, but achieves the highest precision and F1 score. NLSHBlock out-performs NLSHBlock-s because labeled data provides more information on the similarity of item pairs, which is expected. Notably, NLSHBlock-s is also competitive among baselines, even without labeled data, which demonstrates the effectiveness of NLSHLoss. Sparkly is a very strong competitor and outperforms other baselines on the majority of datasets.

We note that for ColBERT, the performance on DS and WA is much lower than that on other datasets because of the size imbalance between TableA and TableB. More specifically, the size of TableB is much larger than TableA, and thus the ratio of matched pairs in the ground-truth for DS and WA is an order of magnitude lower than other datasets. This hinders the model's ability to find similar pairs.

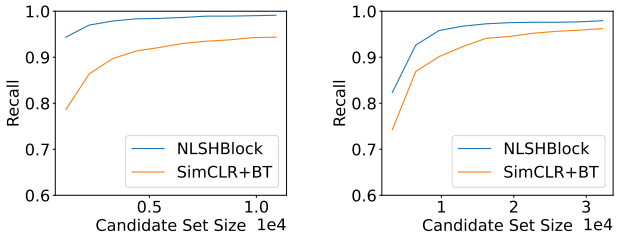
Table 3 lists the candidate sizes of different methods on all datasets. Among all methods, NLSHBlock requires much less candidate pairs to achieve the target recalls on all datasets. This is critical in practice, because the computation cost of the dominating pair-wise matching is significantly reduced. For example, on AB, the candidate set size of NLSHBlock is only 1/2 of the best competitor, which saves about half of the cost.

In summary, given a target recall, NLSHBlock achieves up to 1.95× better F1 score compared to existing best methods, outperforms state-of-the-art methods on a majority of datasets, and only trails the best competitor marginally on the rest of datasets. NLSHBlock can reduce the number of candidate pairs by up to 50% compared to state-of-the-art methods, and thus saves computation cost for matching.

4.5 Ablation Study To understand the effectivenss of our proposed NLSHLoss, we perform an ablation study by disable NLSHLoss in NLSHBlock and only use constrastive loss functions by SimCLR and Barlow Twins (SimCLR+BT) for training the embedding model with the same training examples. As shown in Figure 6, NLSHBlock (blue lines) significantly outperforms SimCLR+BT due to the use of NLSHLoss, which demonstrates the effectiveness of our NLSHLoss in the entity blocking task.

4.6 NLSHBlock for Entity Matching Table 4 shows the performance boost by NLSHBlock on the match-

Figure 6: Efficacy of NLSHLoss on AB and AG



(a) AB (b) AG

ing tasks. We follow the same experimental settings in [47] and do not use labeled data during the training of NLSHBlock for fair comparison with Sudowoodo. The 500 labeled pairs are used in pseudo labeling and the matching model training. On average, NLSHBlock boosts the F1 scores of Sudowoodo by 2.2. Notably, On AB, NLSHBlock boosts the performance of Sudowoodo by up to 5.9 in F1 score. NLSHBlock also outperforms Rotom by a large margin on three datasets and only slightly trails on DS. The performance boost is mainly due to better blocking performance of NLSHBlock, which results in a better quality of probabilistic labels.

Table 4: F1 scores for semi-supervised matching (EM). Ditto, Rotom, Sudowoodo, and NLSHBlock uses 500 uniformly sampled pairs from train+valid.

	AB	AG	DA	DS	WA	average
Ditto	70.1	44.7	95.9	89.4	49.4	69.9
Rotom	69.7	54.0	95.9	91.9	50.1	72.3
Sudowoodo	81.1	59.3	95.2	89.9	66.1	78.3
NLSHBlock	87.0	61.7	97.2	90.7	66.0	80.5
Δ_1 vs Sudowoodo	(+5.9)	(+2.4)	(+2.0)	(+0.8)	(-0.1)	(+2.2)
Δ_2 vs Rotom	(+17.3)	(+17.7)	(+1.3)	(-1.2)	(+16.0)	(+8.2)

5 Conclusion

In this paper, we propose NLSHBlock to approximate locality sensitive hashing functions for finding candidate pairs in entity resolution. NLSHBlock out-performs state-of-the-art methods for the blocking step of the entity resolution task on a wide range of real-world datasets and also boosts the matching performance for the state-of-the-art entity matching method. The key idea of our NLSHBlock method is general and widely applicable. In the future, we will explore the possibility of applying our Neural LSH method on many other tasks such as question answering and recommender systems.

References

[1] M. Akbarian Rastaghi, E. Kamalloo, and D. Rafiei, *Probing the robustness of pre-trained language models for entity matching*, in CIKM, 2022.

[2] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, *Practical and optimal lsh for angular distance*, NeurIPS, 28 (2015).

[3] A. Andoni, A. Naor, A. Nikolov, I. Razenshteyn, and E. Waingarten, *Data-dependent hashing via nonlinear spectral gaps*, in STOC, 2018, pp. 787–800.

[4] A. Andoni and I. Razenshteyn, *Optimal data-dependent hashing for approximate near neighbors*, in STOC, 2015, pp. 793–801.

[5] A. Andoni and I. Razensteyn, *Tight lower bounds for data-dependent locality-sensitive hashing*, in SoCG), vol. 51, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, p. 9.

[6] X. Bai, H. Yang, J. Zhou, P. Ren, and J. Cheng, *Data-dependent hashing based on p-stable distribution*, IEEE Transactions on Image Processing, 23 (2014).

[7] A. Borthwick, S. Ash, B. Pang, S. Qureshi, and T. Jones, *Scalable blocking for very large databases*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2020.

[8] U. Brunner and K. Stockinger, *Entity matching with transformer architectures-a step forward in data integration*, in EDBT, OpenProceedings, 2020.

[9] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, *A simple framework for contrastive learning of visual representations*, in ICML, 2020, pp. 1597–1607.

[10] S. Das, P. S. G. C., and A. D. et. al., *Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services*, in SIGMOD, 2017, pp. 1431–1446.

[11] S. Das, A. Doan, and G. C. et. al., *The magellan data repository.* `https://sites.google.com/site/anhaidgroup/projects/data`.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, *Locality-sensitive hashing scheme based on p-stable distributions*, in SCG, 2004, pp. 253–262.

[13] Y. Dong, P. Indyk, I. P. Razenshteyn, and T. Wagner, *Learning space partitions for nearest neighbor search*, ICLR, (2020).

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

[15] C. Fu, X. Han, J. He, and L. Sun, *Hierarchical matching network for heterogeneous entity resolution*, in IJCAI, 2021, pp. 3665–3671.

[16] J. Gan, J. Feng, Q. Fang, and W. Ng, *Locality-sensitive hashing scheme based on dynamic collision counting*, in SIGMOD, 2012, pp. 541–552.

[17] L. Getoor and A. Machanavajjhala, *Entity resolution: Theory, practice & open challenges*, PVLDB, 5 (2012), pp. 2018–2019.

[18] A. Gionis, P. Indyk, R. Motwani, et al., *Similarity search in high dimensions via hashing*, in VLDB, vol. 99, 1999, pp. 518–529.

[19] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu, *Corleone: hands-off crowdsourcing for entity matching*, in SIGMOD, 2014, pp. 601–612.

[20] P. Indyk and R. Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, in STOC, 1998, pp. 604–613.

[21] G. Izacard, M. Caron, L. Hosseini, S. Riedel, P. Bojanowski, A. Joulin, and E. Grave, *Unsupervised dense information retrieval with contrastive learning*, arXiv:2112.09118, (2021).

[22] J. Johnson, M. Douze, and H. Jégou, *Billion-scale similarity search with gpus*, IEEE Transactions on Big Data, 7 (2019), pp. 535–547.

[23] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, *Dense passage retrieval for open-domain question answering*, arXiv:2004.04906, (2020).

[24] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa, *Low-resource deep entity resolution with transfer and active learning*, in ACL, 2019.

[25] O. Khattab and M. Zaharia, *Colbert: Efficient and effective passage search via contextualized late interaction over bert*, in SIGIR, 2020, pp. 39–48.

[26] P. Konda, S. Das, P. S. G. C., A. Doan, and et al., *Magellan: Toward building entity matching management systems*, PVLDB, 9 (2016).

[27] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan, *Deep entity matching with pre-trained language models*, PVLDB, 14 (2021), pp. 50–60.

[28] Y. Liu, M. Ott, and N. E. A. Goyal, *Roberta: A robustly optimized bert pretraining approach*, arXiv:1907.11692, (2019).

[29] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, *Multi-probe lsh: efficient indexing for high-dimensional similarity search*, in VLDB, Citeseer, 2007, pp. 950–961.

[30] Y. A. Malkov and D. A. Yashunin, *Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs*, IEEE transactions on pattern analysis and machine intelligence, 42 (2018), pp. 824–836.

[31] Z. Miao, Y. Li, and X. Wang, *Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond*, in SIGMOD, 2021, pp. 1303–1316.

[32] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, and et. al., *Deep learning for entity matching: A design space exploration*, in SIGMOD, 2018.

[33] J. Ni, C. Qu, J. Lu, Z. Dai, G. H. Ábrego, J. Ma, V. Y. Zhao, Y. Luan, K. B. Hall, M.-W. Chang, et al., *Large dual encoders are generalizable retrievers*, arXiv:2112.07899, (2021).

[34] H. Nie, X. Han, B. He, L. Sun, B. Chen, W. Zhang, S. Wu, and H. Kong, *Deep sequence-to-sequence entity matching for heterogeneous entity resolution*, in CIKM, 2019, pp. 629–638.

[35] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas, *Blocking and filtering techniques for entity resolution: A survey*, ACM Computing Surveys (CSUR), 53 (2020), pp. 1–42.

[36] 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*, NeurIPS, 32 (2019).

[37] D. Paulsen, Y. Govind, and A. Doan, *Sparkly: A simple yet surprisingly strong tf/idf blocker for entity matching*, Proceedings of the VLDB Endowment, 16 (2023), pp. 1507–1519.

[38] R. Peeters and C. Bizer, *Supervised contrastive learning for product matching*, arXiv:2202.02098, (2022).

[39] R. Peeters, C. Bizer, and G. Glavas, *Intermediate training of BERT for product matching*, in DI2KG@VLDB, F. Piai, D. Firmani, V. Crescenzi, A. D. Angelis, X. L. Dong, M. Mazzei, P. Merialdo, and D. Srivastava, eds., 2020.

[40] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*, Cambridge University Press, 2011.

[41] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, and M. Zaharia, *Colbertv2: Effective and efficient retrieval via lightweight late interaction*, arXiv:2112.01488, (2021).

[42] A. Shrivastava and P. Li, *Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)*, NeurIPS, 27 (2014).

[43] G. Simonini, S. Bergamaschi, and H. V. Jagadish, *BLAST: a loosely schema-aware meta-blocking approach for entity resolution*, PVLDB, 9 (2016).

[44] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi, *Schema-agnostic progressive entity resolution*, TKDE, 31 (2019), pp. 1208–1221.

[45] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. P. G. Fung, and A. Doan, *Blocking in entity matching: A design space exploration*, PVLDB, 14 (2021), pp. 2459–2472.

[46] N. Tonellotto, *Lecture notes on neural information retrieval*, arXiv:2207.13443, (2022).

[47] R. Wang, Y. Li, and J. Wang, *Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation*, ICDE, (2023).

[48] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., *Transformers: State-of-the-art natural language processing*, in EMNLP, 2020.

[49] D. Yao, Y. Gu, G. Cong, H. Jin, and X. Lv, *Entity resolution with hierarchical graph attention networks*, in SIGMOD, 2022, pp. 429–442.

[50] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, in ICML, 2021.

[51] W. Zhang, H. Wei, B. Sisman, X. L. Dong, C. Faloutsos, and D. Page, *Autoblock: A hands-off blocking framework for entity matching*, in WSDM, 2020, pp. 744–752.

[52] K. Zhao, H. Lu, and J. Mei, *Locality preserving hashing*, in AAAI, vol. 28, 2014.

A Supplementary Materials

A.1 Comparisons on Training Data We compare the effect of using different training data for NLSHBlock in Figure 7 and Figure 8. 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 these three settings. On both datasets, using only augmented data leads to the lowest performance, and using both types of data guarantees the best performance.

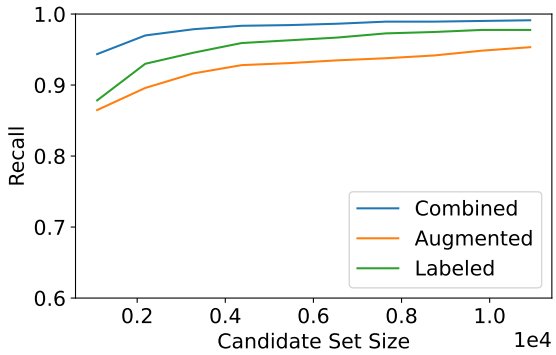


Figure 7: Performance over different training data on AB

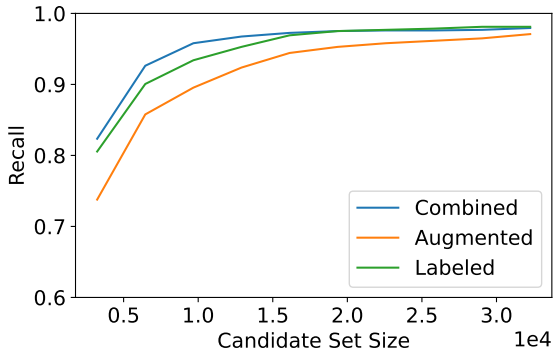


Figure 8: Performance over different training data on AG

A.2 NLSHBlock for Entity Matching - pseudo label quality In Table 5, we report the True Positive Rate (TPR) and True Negative Rate (TNR) of the augmented training set for training entity matching models. The TPR and TNR of pseudo labels generated by NLSHBlock are higher than SimCLR and Sudowoodo on all datasets, which explains the performance gain we observed in Table 4.

A.3 The effect of number of training examples We provide evaluations on the relation between the performance and the number of training tuples. For the results presented in the manuscript, we used 33k and 35k training examples for Abt-Buy and Amazon-Google respectively. In Figure 9 and 10, we compare the performance of NLSH-Block when trained with smaller numbers of training examples. The smaller training sets are randomly sampled from the full training sets. On both datasets, when trained with 5k-35k examples, the performance of NLSH-Block is within a tight band. There is only a slight performance drop when the number of examples is limited to 1k. Besides, only 20% of the training examples are labeled data. This evaluation indicates that NLSHBlock generalizes well when trained with limited labeled data and training examples within the same dataset.

Table 5: True Positive Rate (TPR) and True Negative Rate (TNR) of the training set after adding pseudo labels.

	SimCLR		Sudowoodo		NLSHBlock	
	TPR	TNR	TPR	TNR	TPR	TNR
AB	78.6	97.0	96.4	99.6	98.5	99.6
AG	76.3	96.3	81.8	96.6	85.9	96.8
DA	99.8	98.6	99.8	98.9	1	99.6
DS	99.2	99.5	92.3	98.0	99.9	99.7
WA	69.4	97.0	71.7	97.0	78.7	97.8

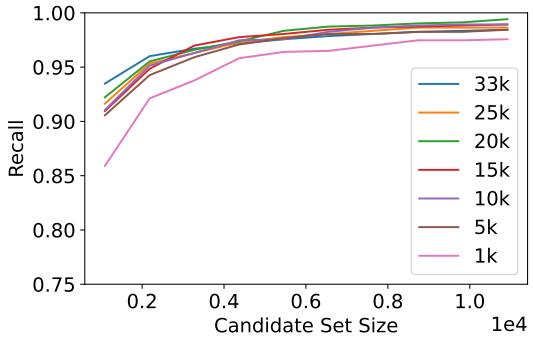


Figure 9: Performance over various training set sizes on AB

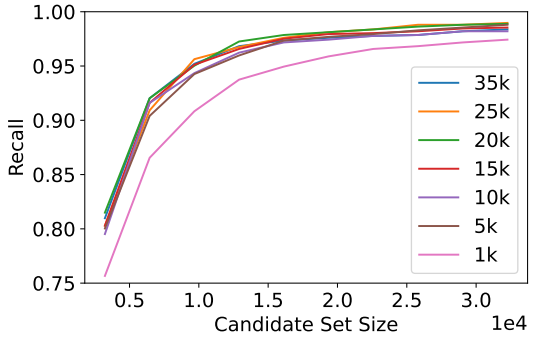


Figure 10: Performance over various training set sizes on AG