SC-Block++: A Blocking Algorithm Based on Adaptive Flood Regularization

Arthur Ning
Erasmus University Rotterdam
Rotterdam, Netherlands
558688an@eur.nl

Flavius Frasincar
Erasmus University Rotterdam
Rotterdam, Netherlands
frasincar@ese.eur.nl

Tarmo Robal
Tallinn University of Technology
Tallinn, Estonia
tarmo.robal@taltech.ee

ABSTRACT

The rapid surge in the number of Web shops presents a challenge for consumers: navigating through the vast amount of stores and products available. Therefore, entity resolution has become an important task to aggregate product information across different Web shops. As entity resolution is a computationally demanding process, its pipelines are divided into two: a blocking phase, which uses a computationally cheap method to select candidate product pairs, and a matching phase with a computationally expensive method to identify matching pairs from the set of candidate pairs. In this paper, we propose SC-Block++, an extension to a state-of-the-art blocking algorithm SC-Block. SC-Block utilizes a RoBERTa base transformer model, trained using Supervised Contrastive Learning, to position the product records in an embedding space, and produces a set of candidate pairs using a nearest-neighbour search. We extend the training procedure of the RoBERTa base transformer model by incorporating Adaptive Flood Regularization (AdaFlood), a regularization method aimed to prevent overfitting and to improve the generalization performance of the model. We compare SC-Block++ to SC-Block, and other benchmark methods on three different data sets, and find that SC-Block++ is able to construct candidate pairs more effectively than the other blocking schemes.

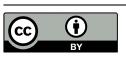
CCS CONCEPTS

• **Information systems** → **Entity resolution**; *Deduplication*; Clustering and classification.

KEYWORDS

Entity resolution; supervised contrastive learning; flooding; adaptive flood regularization

ACM Reference Format:
Arthur Ning, Flavius Frasincar, and Tarmo Robal. 2025. SC-Block++: A Blocking Algorithm Based on Adaptive Flood Regularization. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3672608.3707946



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03.
https://doi.org/10.1145/3672608.3707946

1 INTRODUCTION

The number of online stores is rapidly increasing due to the global digitalization. While these shops may sell identical products, they provide different details about each product. This variation makes it time-consuming for customers to compare the information about a certain product across different Web shops. Therefore, *Entity Resolution* (ER) has become an important task to aggregate product information across different Web shops. These algorithms uncover whether two descriptions, like descriptions of products from two online stores, refer to the same real-world entity. An entity resolution pipeline typically consists of the following two steps: *blocking* and *matching*. Blocking methods are computationally cheap methods that construct *candidate pairs*, pairs of entities that are most likely to match, in order to reduce the number of pairs that need to be compared by the matching algorithm. On the other hand, matching schemes are computationally more expensive than blocking algorithms and try to identify the candidate pairs that refer to the same real-world entity.

In this paper, we focus on the state-of-the-art blocking method *SC-Block* from Brinkmann et al. [4]. This method applies *Supervised Contrastive (SupCon) Learning* to train a *RoBERTa base language model* to position records describing the same real-world entity close to each other in an embedding space. Unfortunately, neural networks (NN) like the RoBERTa base transformer model are prone to overfitting. When we train the model for a long time, we can easily achieve a model with a *training loss* (value of the objective function we minimize) of 0 after acquiring a *training error* (fraction of training samples incorrectly fitted by the model) of 0. Recent works have shown that it is important to train a model until it achieves a training error close to 0 to decrease its generalization error [3, 14]. Yet, Ishida et al. [8] are uncertain whether a training loss of 0 is necessary after achieving a training error of 0. Namely, a model with a training loss equal to 0 may be fitted too well to the training data, which degrades its performance to make predictions on unseen data.

For the above reason, [8] proposes a method known as *Flooding*, which keeps the training loss at a pre-specified value close to 0, known as the *flood level*, showing evidence that Flooding is effective at mitigating overfitting. Several extensions to Flooding have been proposed. One of these is *Adaptive Flood Regularization (AdaFlood)* [2], which in contrast to the regular Flooding procedure takes into account the training difficulty of each training sample by computing a flood level for each sample. Subsequently, the individual loss of each training sample is kept at their respective flood

level. Bae et al. [2] show that AdaFlood is able to consistently outperform Flooding. Hence, we introduce *SC-Block++*, a new variant of SC-Block, where we incorporate AdaFlood during the training procedure of the RoBERTa base transformer model. We show that AdaFlood is able to enhance the ability of SC-Block to construct candidate pairs effectively. The Python code of the project is made freely available at https://github.com/arthrng/scblock-plusplus.

We structure the rest of the paper as follows. In Section 2, we provide an overview of the literature regarding blocking methods and regularization techniques. Section 3 discusses how SC-Block, Flooding, and AdaFlood work, and how our proposed method SC-Block++ makes use of these. Section 4, presents the data we use to evaluate the performance of SC-Block++, and Section 5 performance evaluation of SC-Block++ against other blocking methods. Last, in Section 6, we conclude and provide suggestions for future research.

2 RELATED WORK

In this section, we describe work related to ours. We start with the blocking methods, and then discuss regularization methods used to prevent overfitting.

Blocking algorithms aim to find candidate pairs of descriptions which are most likely to refer to the same real-world entity. This is done in order to decrease the number of comparisons that need to be performed by the matching methods in ER pipelines. The easiest way to block entities when the schema of the data set is available, is to group the records based on their value(s) for some selected key(s) [5]. For example, one can group records of people in a database based on their postal code, which greatly reduces the comparisons required to find which records refer to the same person.

Many methods build upon the idea that candidate pairs can be found by grouping records in a database based on some selected keys. Aizawa and Oyama [1] propose *suffix array blocking*. This method extracts the suffixes with some predetermined minimum length from certain keys. Subsequently, a block is constructed for each suffix in which the records sharing the same suffix for the selected keys are stored.

Recently, deep learning for blocking has become popular. *AutoBlock* is one of the earliest frameworks to use supervised deep learning for blocking [27]. Similarly to SC-Block, it utilizes a language model to construct embedding vectors for the records. Next, *Locality-Sensitive Hashing (LSH)* is used to retrieve the nearest neighbours of each record such that candidate pairs can be constructed. The shortcoming of AutoBlock is that it requires labelled data. In other words, each record in the data set needs a label to detect which samples refer to the same real-world object. In reality, it may be difficult to acquire labelled data. Hence, Thirumuruganathan et al. [21] introduce an unsupervised blocking framework called *DeepBlocker*, offering multiple deep learning-based blocking solutions. These solutions exploit sequence modelling, transformer, and self-supervision. Afterwards, one can utilize a similarity-based, hash-based, or composite approach to construct candidate pairs based on the embeddings.

In the field of supervised machine learning, overfitting is a fundamental issue due to the complexity of the model, the presence of noisy data in the training set, or the limited number of samples in the training data [22]. A machine learning model may fit very well to the training data, but it can perform very poorly when making predictions on unseen data as a result of overfitting. Over the years, many regularization methods have been developed in order to mitigate overfitting.

One of the earliest and most trivial regularization methods to reduce the complexity of a NN is *weight decay*, introduced by Hanson and Pratt [6], which aims to reduce the size of the parameters in the model by adding a penalty term to the original objective function. Nevertheless, Krizhevsky et al. [11] find that weight decay can reduce overfitting more effectively when combined with *dropout*, a technique by Hinton et al. [7]. The key idea of dropout is that nodes in the NN are randomly dropped with probability p and retained with probability $1 - p$ in each batch of the training process to reduce the complexity of the NN. If a node is dropped, its incoming and outgoing edges are also removed from the network. Repeatedly dropping nodes each epoch results in many variants of the original NN, which are each trained. Since all these variants share their parameters with the original NN, we are simultaneously training the original network by training the smaller variants of it. To compensate for the reduction in the size of the parameters, in the final batch we scale the parameters with a factor $1 - p$. Srivastava et al. [20] find that overfitting can be reduced and the predictive performance of NNs improves if dropout is used.

Another regularization method is *early stopping* [25]. When training a deep learning model, the training and validation loss typically decrease with more iterations. However, too many iterations can lead to the validation loss increasing again. Thus, in the early stopping procedure we stop training the model once there is an increase of the validation loss.

Ishida et al. [8] state that the regularization methods cannot directly control the training loss. Hence, although it becomes more difficult for the training loss to shrink towards 0 when using these techniques, it is hard to maintain a training loss greater than 0 until the training procedure is finished. Thus, there is still some risk that a model may overfit. Furthermore, some of these regularization methods are specific to a model or a task (e.g., weight decay and dropout are regularization methods tailored to NNs) [8].

For the given reasons, Ishida et al. [8] introduce Flooding, a regularization method which directly prevents the training loss from decreasing when it reaches a reasonably small value chosen by a researcher. Also, this method is not limited to a specific task or type of model. Therefore, in this research, we utilize Flooding to improve the performance of SC-Block.

3 METHODOLOGY

In Section 3.1, we introduce the SC-Block blocking method, then in Section 3.2, we discuss Flooding, and last in Section 3.3, we present AdaFlood and how we can incorporate it in the training procedure of SC-Block.

For the remainder of the paper, we assume that we have two sets $\mathcal{D}$ and $\mathcal{E}$ containing a number of product offers such that $|\mathcal{E}| < |\mathcal{D}|$. Both data sets are split into training, validation, and test sets, which denoted as $\mathcal{D}_{train}$ and $\mathcal{E}_{train}$, $\mathcal{D}_{val}$ and $\mathcal{E}_{val}$, and $\mathcal{D}_{test}$ and $\mathcal{E}_{test}$, respectively. Furthermore, we assume that the records in both data sets follow the same schema. Lastly, we assume that we have a *Clean-Clean* entity resolution problem, meaning that $\mathcal{D}$ and $\mathcal{E}$ individually do not contain any duplicate product offers [16].

3.1 SC-Block

SC-Block is a blocking method by Brinkmann et al. [4] that combines SupCon Learning for positioning records in an embedding space with a nearest-neighbour search to construct candidate product pairs. In this section, we give an overview of how this method is used to find candidate pairs, with steps performed by SC-Block illustrated in Figure 1.

3.1.1 Preparing the Data. The product offers i in the sets $\mathcal{D}$ and $\mathcal{E}$ have a title t_i and certain attributes which are stored in the key-value pairs $k_1, v_{1,i}, \ldots, k_p, v_{p,i}$, where k_j is the name of attribute j, $v_{j,i}$ is the value of attribute j of product offer i, and p is the number of attributes that the records in both $\mathcal{D}$ and $\mathcal{E}$ have. For each product offer, we serialize their title and attributes to a single string s_i as follows:

s_i = "$COL\ title\ VAL\ t_i\ COL\ k_1\ VAL\ v_{1,i}\ \ldots COL\ k_p\ VAL\ v_{p,i}$".

The product offers in some of the data sets we use are stored in pairs i, j, where $i \in \mathcal{D}$ and $j \in \mathcal{E}$. Each pair has a label $l_{i,j}$ such that $l_{i,j} = 1$ if i and j refer to same real-world entity and $l_{i,j} = 0$ otherwise. The SupCon loss we use to train the RoBERTa base transformer model requires that each product offer i in the training set has its own label l_i, such that matching product offers share the same label. To obtain these labels, we build a correspondence graph using the procedure described by Peeters and Bizer [17]. Each vertex in this correspondence graph represents a product offer from either $\mathcal{D}_{train}$ or $\mathcal{E}_{train}$. We use the labels $l_{i,j}$ to connect the vertices representing the matching product offers i and j. After connecting all the matching product descriptions, we can assign a unique label to each vertex in the graph such that matching records share the same label.

3.1.2 Creating Embeddings. In each epoch of the training procedure, we construct the batches $\mathcal{B}_1, \ldots, \mathcal{B}_B$ containing 128 strings corresponding to individual product offers (not pairs of products), which are all randomly sampled from either $\mathcal{D}_{train}$ or $\mathcal{E}_{train}$. As we use the Supervised Contrastive Loss to train the RoBERTa base transformer model, we require a *multiviewed batch* [10]. A multiviewed batch $\tilde{\mathcal{B}}_k$, $k = 1, \ldots, B$, contains the samples from $\mathcal{B}_k$ and an augmented version of these samples. To create the multiviewed batch, for each string s_i in $\mathcal{B}_k$, we find another string s_j pertaining to a product offer, which has the same label as i, and store it in the multiviewed batch. We sample s_j from $\mathcal{D}_{train}$ if $i \in \mathcal{E}_{train}$, otherwise we sample s_j from $\mathcal{E}_{train}$. If there does not exist a product offer with the same label as i, we simply duplicate i and store this duplicate in the multiviewed batch. Thus, each multiviewed batch contains 256 product offers. In each epoch, we construct batches until every serialized product string has been in some batch $\mathcal{B}_k, k = 1, \ldots, B$.

We utilize the RoBERTa base transformer model to create embeddings of the serialized strings, numerical representations of the strings, expressed as a vector. We perform a nearest neighbour search on these embeddings which is used to construct the candidate pairs. To generate the embeddings for i and j, we provide s_i and its "augmented" variant s_j from the multiviewed batch as the input to the RoBERTa base transformer model, which constructs embeddings for each token in both strings. After we obtain the embeddings for i and j, we *mean-pool* and normalize them using an L_2 normalization such that we obtain the vectors $\hat{\boldsymbol{z}}_i$ and $\hat{\boldsymbol{z}}_j$, $||\hat{\boldsymbol{z}}_i||_2 = ||\hat{\boldsymbol{z}}_j||_2 = 1$.

To update the parameters in the RoBERTa base transformer model during the backpropagation step of the training procedure, we use the SupCon loss function denoted as:

$$\ell_{SupCon} = \sum_{i \in \tilde{\mathcal{B}}_k} -\frac{1}{|Pi|} \sum_{p \in Pi} \ln\Big(\frac{\exp \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_p \mathcal{T}}{\sum_{a \in Ai} \exp \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_a \mathcal{T}}\Big), \quad (1)$$

where $Pi = \{p \in \tilde{\mathcal{B}}_k | l_p = l_i\}$, $Ai = \tilde{\mathcal{B}}_k \backslash \{i\}$, $\mathcal{T} \in \mathbb{R}$ is a scalar temperature parameter, and $\hat{\boldsymbol{z}}_i$ is denoted as the anchor embedding for product offer i. This loss function tries to maximize the agreement of records with the same label and minimizing the agreement of records with different labels. Hence, we try to pull the anchor embedding of each product $i \in B_k$ closer to the other embeddings with the same label while pushing it away from embeddings with different labels.

To train the RoBERTa model, we utilize the experimental setup of Brinkmann et al. [4], setting $d = 768$, $\mathcal{T} = 0.07$, and we train the model for 20 epochs using the Adam optimizer with a learning rate of 5×10^{-5}. Yet, the batch size we use is different. Namely, [4] uses a batch size of 512, while we use a size of 128 due to limitations of our computational resources. This smaller batch size may hamper the performance of SC-Block, as Peeters and Bizer [17] state that the SupCon loss benefits from having a large number of samples in a batch.

3.1.3 Finding Candidate Pairs. After we train the RoBERTa base transformer model, we embed the product offers from $\mathcal{D}_{test}$ and $\mathcal{E}_{test}$ to the embedding space. We use the embeddings from $\mathcal{D}_{test}$ to construct the query table $\mathcal{Q}$ and the embeddings from $\mathcal{E}_{test}$ to create the index table $\mathcal{I}$. The index records in $\mathcal{I}$ are constructed using *FAISS* Python library [9], to allow for an efficient searching procedure.

Once we have prepared the tables $\mathcal{Q}$ and $\mathcal{I}$, we iterate over all the embeddings in $\mathcal{Q}$. For each $\boldsymbol{z}_a \in \mathcal{Q}$, we compute its cosine similarity with each embedding $z_b \in \mathcal{I}$ (Eq. 2),

$$\cos \hat{\boldsymbol{z}}_a, \hat{\boldsymbol{z}}_b = \frac{\hat{\boldsymbol{z}}_a \cdot \hat{\boldsymbol{z}}_b}{||\hat{\boldsymbol{z}}_b||\ ||\hat{\boldsymbol{z}}_b||}, \quad (2)$$

where $\cos \hat{\boldsymbol{z}}_a, \hat{\boldsymbol{z}}_b \in -1, 1$. The closer $\cos \hat{\boldsymbol{z}}_a, \hat{\boldsymbol{z}}_b$ is to 1, the more semantically similar the two embeddings are to each other. We order the embeddings in $\mathcal{I}$ using merge sort based on their cosine similarity with $\hat{\boldsymbol{z}}_a$, which has a time complexity of $\mathcal{O}n \log_2 n$ in the worst-case possible case. Thereafter, we select the k embeddings $\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_k \in \mathcal{I}$ which have the highest value

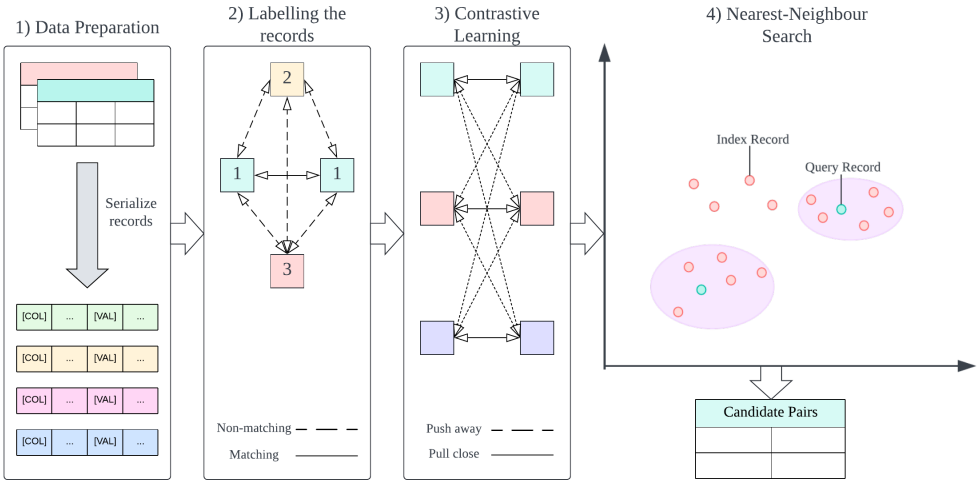


Figure 1: Overview of SC-Block.

for the cosine similarity with $\hat{\boldsymbol{z}}_a$. If $a \in \mathcal{D}_{test}$ is the product that belongs to embedding $\hat{\boldsymbol{z}}_a$ and $b_1, \ldots, b_k \in \mathcal{E}_{test}$ are the products that correspond to $\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_k$, then we denote the candidate pairs as $a, b_1, \ldots, a, b_k$.

3.2 Flooding

Flooding is a technique proposed by Ishida et al. [8] that prevents a deep neural network from overfitting on the training data. Let $\mathcal{S} = \{\boldsymbol{x}_i, y_i | \boldsymbol{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1, \ldots, N\}$ be the training set and $f : \mathcal{X} \to \mathcal{Y}$ the NN we are training. Furthermore, suppose that ℓ is some arbitrary loss function and $\mathcal{L}_{train} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell y_i, f\boldsymbol{x}_i$ is the training loss of our model, where $\mathcal{B}$ is a batch containing samples from $\mathcal{S}$. Last, we assume that $\boldsymbol{\theta}$ contains the parameters in f that we want to estimate. Then, we can constraint the training loss by some lower bound $\gamma > 0$, also known as the flood level, using

$$\tilde{\mathcal{L}}_{train}\boldsymbol{\theta} = |\mathcal{L}_{train}\boldsymbol{\theta} - \gamma| + \gamma. \tag{3}$$

This expression ensures that the gradient of $\tilde{\mathcal{L}}_{train}$ w.r.t. $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_{train}$, points in the same direction as $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{train}$, when $\mathcal{L}_{train}\boldsymbol{\theta} > \gamma$, but in the opposite direction when $\mathcal{L}_{train}\boldsymbol{\theta} < \gamma$. This results in the training loss floating around γ.

3.3 Adaptive Flooding Regularization

3.3.1 The Method. AdaFlood is regularizer proposed by Bae et al. [2] based on Flooding, which tries to mitigate two shortcomings of Flooding. First, Flooding may lead to instability issues – training a machine learning model with Flooding multiple times can lead to different solutions of the parameter estimates, each different in their ability to generalize. This instability issue stems from the fact that the training procedure of the RoBERTa base transformer model is stochastic, as we construct batches by randomly sampling products from the full training set. Furthermore, Flooding only guarantees "global convergence" —Flooding encourages the training loss $\mathcal{L}_{train}$, averaged over all the samples in a batch, to be close to γ, while having no requirement on the individual losses of the training samples. As shown empirically by Xie et al. [24], the individual losses of some training samples are far away from γ while others are close to γ, leading to some of the inconsistent behaviour of Flooding.

Second, Flooding does not take into account the training difficulty of the samples. For example, some samples are "easier" to classify for models, meaning that their individual training loss can be driven to 0 without overfitting the model. However, for outliers, noisy, or incorrectly labelled training samples, a training loss of 0 may cause overfitting.

Bae et al. [2] mitigate both shortcomings of Flooding by reformulating Eq. 3 to

$$\tilde{\mathcal{L}}_{AdaFlood,train}\boldsymbol{\theta} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} |\ell y_i, f\boldsymbol{x}_i - \gamma_i| + \gamma_i, \tag{4}$$

where γ_i is the flood level belonging to some training sample i, which addresses the second issue we discuss. This new formulation solves the first issue of Flooding by subtracting the flood level from each individual training loss, rather than the training loss $\mathcal{L}_{train}$ computed over the entire batch. Therefore, we aim to keep the individual losses of each training sample close to their respective flood levels, leading to more consistent behaviour of the models.

We compute the individual flood level for each training sample before we train the NN f using Eq. 5, where $f_{i,j,aux}$ is some arbitrary auxiliary model, $j = 1, \ldots H$, $H \leq N$.

$$\gamma_i = \ell y_i, f_{i,j,aux}\boldsymbol{x}_i, \tag{5}$$

3.3.2 Adaptive Flooding within SC-Block. To incorporate AdaFlood within the SC-Block framework, resulting in our proposed method SC-Block++, we train some auxiliary networks. We initiate the estimation procedure by training a RoBERTa base transformer model on $\mathcal{D}_{train} \cup \mathcal{E}_{train}$ using the exact same procedure described in Sections 3.1.1 and 3.1.2. We label this model as the base auxiliary network f_{aux}.

Then, we partition $\mathcal{D}_{train} \cup \mathcal{E}_{train}$ into H equal-sized, non-overlapping sets $\mathcal{D}_{1,train} \cup \mathcal{E}_{1,train}, \ldots, \mathcal{D}_{H,train} \cup \mathcal{E}_{H,train}$, where each set contains n samples (although the size of $\mathcal{D}_{H,train} \cup \mathcal{E}_{H,train}$ may be smaller than n if it is not possible to precisely divide $\mathcal{D}_{train} \cup \mathcal{E}_{train}$ into H equal-sized sets). For $i \in \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$ and $j = 1, \ldots, H$, we train an auxiliary network $f_{i,j,aux}$ by first randomly re-initializing the weights of the last three layers of f_{aux}. Namely, Bae et al. [2] state that re-initializing the weights of the last few layers removes most of the influence of the samples from $\mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$ on $f_{i,j,aux}$. Subsequently, we train $f_{i,j,aux}$ using $\mathcal{D}_{train} \cup \mathcal{E}_{train} \setminus \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$. Again, we use the procedure from Sections 3.1.1 and 3.1.2 to train the auxiliary network. Once we have trained $f_{i,j,aux}$, we utilize $f_{i,j,aux}$ to determine the mean-pooled and normalized embeddings $\hat{\boldsymbol{z}}_1, \ldots, \hat{\boldsymbol{z}}_n$ belonging to the product offers in $\mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$. Last, we can determine the flood level γ_i by computing the individual SupCon loss for the product offer $i \in \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$, which we denote as $\ell_{i,SupCon}$. We compute $\ell_{i,SupCon}$ using

$$\ell_{i,SupCon} = -\frac{1}{|Pi|} \sum_{p \in Pi} \ln\Big(\frac{\exp \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_p \mathcal{T}}{\sum_{a \in Ai} \exp \hat{\boldsymbol{z}}_i \cdot \hat{\boldsymbol{z}}_a \mathcal{T}}\Big), \quad (6)$$

where $Pi = \{p \in \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train} | l_p = l_i\}$, $Ai = \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train} \setminus \{i\}$, and the rest of the notation is the same as defined in Section 3.1.2. We repeat this procedure until each product offer $\mathcal{D}_{train} \cup \mathcal{E}_{train}$ has its own flood level.

After we have determined the flood levels for all the samples in $\mathcal{D}_{train} \cup \mathcal{E}_{train}$, we can use the training procedure from Section 3.1.2 to train the RoBERTa base transformer model. However, we utilize $\tilde{\mathcal{L}}_{AdaFlood,train}$ from Eq. 4 as the loss function, where each individual loss of the training samples are computed using Eq. 6.

We estimate $H = 20$ auxiliary NNs. This implies that the data we use to train $f_{i,j,aux}$, $\mathcal{D}_{train} \cup \mathcal{E}_{train} \setminus \mathcal{D}_{j,train} \cup \mathcal{E}_{j,train}$, contains roughly 95% of the product offers from $\mathcal{D}_{train} \cup \mathcal{E}_{train}$. According to [2], this size for the training set of the auxiliary networks should roughly approximate the best performance we can achieve using AdaFlood. Furthermore, we train each auxiliary network for 20 epochs using the Adam optimizer with a learning of rate 5×10^{-5}. The embeddings constructed by the auxiliary networks have size $d = 768$, the temperature parameter $\mathcal{T}$ in the SupCon loss function is set to 0.07, and the batches in each epoch have size 128.

4 DATA

We use three data sets for our research to compare the various blocking methods to SC-Block.

First, we use is the Web Data Commons (WDC) Products data set[1] introduced by Peeters et al. [18] to evaluate blocking methods. This data set contains 11715 product offers describing 2162 unique real-world products. These product offers have been extracted from 3259 Web shops in 2020 using the schema.org annotations. Each product offer is described using the following 5 attributes: title, description, price, currency, and brand. The data set also contains pairs of product offers which have a label of 1 if they match and 0 otherwise. Yet, the data set is not exhaustive, meaning that it does not include all possible product pairs, as labelling all the pairs may be computationally expensive.

Table 1: Statistics of the WDC products data sets

Split	Corner cases	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{E}\rvert$	# entities in $\mathcal{D} \cup \mathcal{E}$ without duplicates	# matching pairs	# non-matching pairs
Training	20%	500	894	309	500	2000
Validation		500	891	306	500	2000
Test		500	719	147	500	4000
Training	80%	500	881	278	500	2000
Validation		500	894	304	500	2000
Test		500	711	156	500	4000

To evaluate the performance of the blocking methods, *corner cases* are included in the data set. There are two types of corner cases: positive and negative corner cases. A positive corner case refers to matching products whose textual representations are highly dissimilar from each other. For example, some online stores mention different features or use different units of measurements to describe the same product. A negative corner case is given by two product offers which do not match but whose descriptions exhibit many similarities. Thus, two product offers which differ in only one product feature are considered a negative corner case.

The WDC Products data set provides more challenges to the blocking methods by including product descriptions of real-world entities in the test set which are not included in the training data. Peeters and Bizer [17] state that addition of unseen product entities in the test set allows to assess the robustness of the methods regarding their performance matching unseen entities.

Multiple variations of the WDC data set varying in size, the number of corner cases, and the number of unseen products in the test set exist. We exploit the "small" variant of the WDC Products data set, containing a total of 6500 product offers. Furthermore, we select the variants of this small data set in which 20% and 80% of the product offers are corner cases (further referred to as WDC_{20} and WDC_{80} correspondingly) to measure how well the blocking methods are able to deal with difficult product offers. Last, 50% of the product offers in the test set of both variants of the small data set are unseen in the training data. Table 1 provides some statistics of these two data sets.

Second, for the training and evaluation of the blocking methods in this paper, we also consider a data set known as Walmart-Amazon[2]. This data set is provided by Mudgal et al. [13] for evaluating entity resolution methods. The data set contains a total of 24628 product offers for electronics collected from the Web shops Amazon (22074 product offers) and Walmart (2554 product offers). Each product offer has the following 5 attributes: title, product category, brand, model number, and price.

[1] Downloaded from https://webdatacommons.org/largescaleproductcorpus/wdc-block on April 25th 2024.

[2] Downloaded from https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md on April 25th 2024.

Table 2 provides the statistics of the Walmart-Amazon data set, where products from Walmart are denoted as $\mathcal{D}$ and the ones from Amazon as $\mathcal{E}$. We observe that the number of pairs in each split is not close to the size of the Cartesian product $\mathcal{D} \times \mathcal{E}$ (i.e., $22074 \times 2554 = 56376996$), which we also observe in the previous data set. Again, labelling each pair may be computationally expensive. Therefore, Mudgal et al. [13] only consider a subset of the pairs from $\mathcal{D} \times \mathcal{E}$.

Table 2: Statistics of the Walmart-Amazon data set

Data set	Split	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{E}\rvert$	# entities in $\mathcal{D} \cup \mathcal{E}$ without duplicates	# matching pairs	# non-matching pairs
Walmart-Amazon	Training	1424	3702	3742	576	5568
	Validation	927	1580	1756	193	1856
	Test	900	1584	1770	193	1856

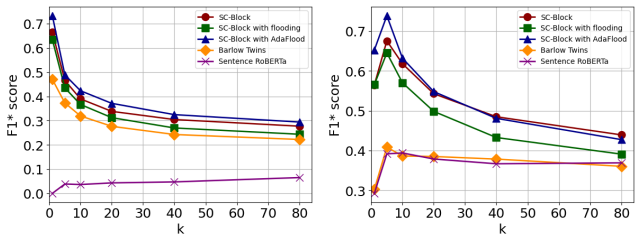
5 RESULTS

In this section, we evaluate the performance of SC-Block++, first by comparing it to SC-Block, SC-Block with Flooding (SC-Block$_{Flooding}$), another supervised blocking method known as *Sentence-RoBERTa (SRoBERTa)* [19], and a self-supervised blocking method called *Barlow Twins* [26] (Section 5.1). Next, we evaluate the performance of the blocking methods within entity resolution pipelines in Section 5.2. We carry out all experiments using an NVIDIA A100 as our GPU.

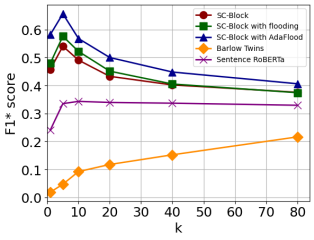
5.1 Comparing the Blocking Methods

We apply the blocking methods SC-Block, SC-Block$_{Flooding}$ (we provide the optimal values of γ in Appendix A), SC-Block++, SRoBERTa, and Barlow Twins to the test splits of all three data sets for $k \in \{1, 5, 10, 20, 40, 80\}$ nearest neighbours. Subsequently, we compute the *Pair Quality*, *Pair Completeness*, and F_1^* *score* to evaluate these methods. Pair Quality reflects the fraction of true matches that are maintained after blocking. Pair Completeness measures the fraction of pairs that are correctly identified as candidate pairs. F_1^* score is the harmonic mean between the Pair Quality and Pair Completeness, and allows us to measure how well a blocking method is able to balance these two metrics.

In Figures 2, 3, and 4 we report the F_1^* score, Pair Quality and Pair Completeness obtained by the blocking methods versus k, respectively. From the figures we observe that, overall, SC-Block++ outperforms all other blocking methods. Specifically, SC-Block++ achieves a higher F_1^* score across all data sets and most values of k compared to the other blocking methods, implying that SC-Block++ is able to achieve a better balance between Pair Quality and Pair Completeness. Indeed, if we look at the Pair Quality we see that, generally, the Pair Quality of SC-Block++ is larger than that of the other blocking methods. This implies that a larger proportion of the pairs in the candidate set constructed by SC-Block++ are truly matching, implying that SC-Block++ is more effective at finding truly matching pairs. Furthermore, we observe that the Pair Completeness of SC-Block++ is generally larger compared to the other blocking methods, and therefore approaches 1 much sooner. This means that

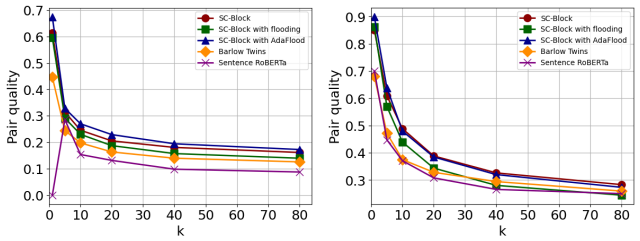


(a) Walmart-Amazon **(b) WDC$_{20}$**

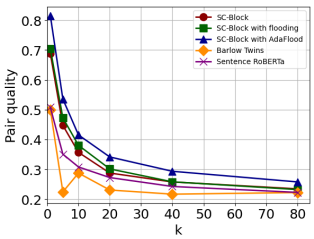


(c) WDC$_{80}$

Figure 2: F_1^* score of each blocking method.



(a) Walmart-Amazon **(b) WDC$_{20}$**

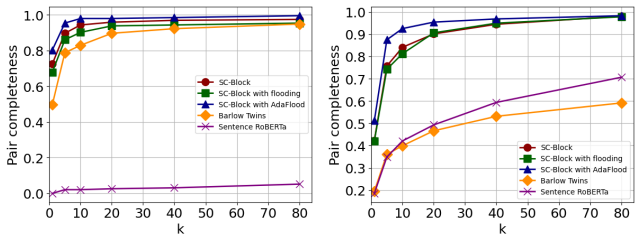


(c) WDC$_{80}$

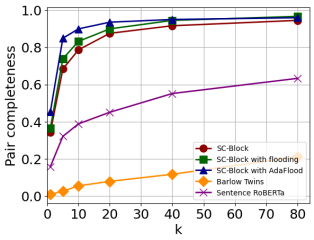
Figure 3: Pair Quality of each blocking method.

SC-Block++ already captures many of the matching pairs for a smaller value k, reducing the number of comparisons that need to be done by the matching method. From these findings, it is evident that SC-Block benefits from AdaFlood.

Nevertheless, we do not obtain the same success for SC-Block$_{Flooding}$.Namely, SC-Block$_{Flooding}$ achieves much lower F_1^* scores compared to SC-Block when applied to the data sets Walmart-Amazon and WDC$_{20}$. These findings are also reflected in the Pair Completeness and Pair Quality of SC-Block$_{Flooding}$ being lower than those of SC-Block for these two data sets. We believe that these results may be caused



(a) Walmart-Amazon **(b) WDC$_{20}$**



(c) WDC$_{80}$

Figure 4: Pair Completeness of each blocking method.

by the instability issues that Flooding has, which we discussed in Section 3.3. Hence, the weights in the RoBERTa base transformer model of SC-Block$_{Flooding}$ that we obtain may not result in the best performance on the test set. Also, we may obtain these results because Flooding simply is not necessary when applying SC-Block to Walmart-Amazon and WDC$_{20}$ (optimal level of γ is 0 as shown in Appendix A).

When we look at the results of SC-Block$_{Flooding}$ for WDC$_{80}$, SC-Block$_{Flooding}$ is able to outperform SC-Block. Specifically, the F_1^* scores of SC-Block$_{Flooding}$ are strictly larger than the scores of SC-Block for $k = 1, 5, 10$, and 20. Namely, for these values of k, SC-Block$_{Flooding}$ manages to obtain higher values of the Pair Quality and Pair Completeness. These results may confirm our finding from Appendix A: Flooding can help SC-Block deal with the large amount of corner cases in the WDC Products data set.

5.2 Comparing the Blockers within ER Pipelines

To compare how blocking methods perform within ER pipelines, we first tune k for the blocking methods across all three data sets. To tune k, we utilize the procedure used by Brinkmann et al. [4] and Papadakis et al. [15]. Namely, for each blocking method, we compute the Pair Completeness on the validation split of the data sets for the values $k \in \{1, \ldots, 80\}$ until it reaches a threshold of 95%. Once the Pair Completeness reaches this threshold, we find the optimal value of k, as it means that we have captured most of the matching pairs in the candidate set. If a blocking method is not able to reach the threshold of 95%, we simply set $k = 80$. The rationale for tuning k based on the Pair Completeness instead of the F_1^* score stems from the fact that, during the matching phase, any matching pair discarded by the blocking method cannot be recovered. Therefore, we tune k in such a way that we add most of the matching pairs to the candidate set while keeping the number of candidate pairs as small as possible.

After we find the optimal k for SC-Block, SC-Block$_{Flooding}$, SC-Block++, SRoBERTa, and Barlow Twins across all three data sets, we incorporate these blocking methods within ER pipelines, with the matching methods being *SupCon-Match* [17], *Ditto* [12], and *DeepMatcher* [13]. We use the ER pipelines to extract matching pairs from the test split of all three data sets. We report for the ER pipelines the optimal value of k obtained by the blocking method, the Fraction Of Comparisons (FOC), precision, recall, and F_1 score in Table 3. The FOC is defined as $\frac{|\mathcal{C}|}{|\mathcal{D}_{test}| \times |\mathcal{E}_{test}|}$, where $\mathcal{C}$ is the candidate pairs set in the test set. The optimal values of the hyperparameter α used in Ditto are 0.4 for Walmart-Amazon, 0.6 for WDC$_{20}$, and 0.1 for WDC$_{80}$.

From Table 3, we observe that across all three data sets, the pipelines that do not contain a blocking scheme achieve the best performance in terms of precision, recall, and the F_1 score. In terms of efficiency, the pipelines without a blocking method perform the worst since the matching schemes need to consider every possible comparison. In contrast, the pipelines that use SC-Block++ as the blocking scheme are able to find a better balance between the efficiency of the matching method and the quality of the pairs constructed by the matching method. Namely, we observe that the SC-Block++ pipelines obtain values of the precision, recall, and the F_1 score similar to the ones obtained by the pipelines without a blocking scheme, while requiring less comparisons.

In addition, the SC-Block++ pipelines outperform the other pipelines where blocking is performed. The SC-Block++ manages to reach the Pair Completeness threshold of 95% for a much smaller value k compared to the other blocking methods, resulting in the matching methods within SC-Block++ pipelines performing less comparisons, as is also evident from the FOC results. Although the pipelines with SC-Block++ do not always obtain the highest recall, especially on the WDC$_{80}$ data set, their F_1 scores are larger compared to the other pipelines since the pipelines are more precise. The higher precision of the SC-Block++ pipelines is attributed to the fact that SC-Block++ constructs much smaller candidate sets than the other blocking schemes, as k is smaller. Therefore, the candidate set of SC-Block++ contains less false positives, increasing the precision of the SC-Block++ pipelines relative to the other pipelines.

When we look at the pipelines that use SC-Block$_{Flooding}$, we notice that the SC-Block$_{Flooding}$ pipelines outperform the SC-Block pipelines on the WDC$_{80}$ data set. Specifically, the SC-Block$_{Flooding}$ pipelines are able to achieve a lower value of k, leading to less product pairs that need to be compared by the matching method. Furthermore, this reduction in the number of comparisons leads to a higher precision, and thus a higher F_1 score. These results are in line with the notion from the previous section that SC-Block$_{Flooding}$ is able to improve the tolerance of SC-Block against corner cases. Still, on the two other data sets, the SC-Block$_{Flooding}$ pipelines generally perform worse than the SC-Block pipelines, in terms

Table 3: Performance of the ER pipelines for Walmart-Amazon, WDC_{20} and WDC_{80}. The best value of a measure is in bold

		Walmart-Amazon					WDC_{20}					WDC_{80}				
Blocker	Matcher	k	FOC	Recall	Precision	F_1	k	FOC	Recall	Precision	F_1	k	FOC	Recall	Precision	F_1
-	SupCon-Match	-	100%	62.2%	**58.8%**	**60.4%**	-	100%	75.6%	**64.9%**	**69.8%**	-	100%	72.1%	**60.4%**	**65.7%**
	Ditto	-	100%	97.0%	34.5%	50.9%	-	100%	93.1%	48.5%	63.8%	-	100%	92.5%	45.1%	60.6%
	DeepMatcher	-	100%	**98.1%**	28.3%	43.9%	-	100%	**96.1%**	41.2%	57.7%	-	100%	94.0%	35.1%	51.1%
SC-Block++	SupCon-Match	**7**	**0.8%**	57.6%	54.9%	56.2%	**15**	**2.1%**	73.3%	63.1%	67.8%	**10**	**2.0%**	70.0%	59.1%	64.1%
	Ditto	**7**	**0.8%**	95.3%	32.6%	48.9%	**15**	**2.1%**	92.5%	47.9%	63.2%	**10**	**2.0%**	89.9%	41.5%	56.8%
	DeepMatcher	**7**	**0.8%**	97.9%	27.0%	42.3%	**15**	**2.1%**	95.4%	38.4%	54.8%	**10**	**2.0%**	93.5%	34.2%	50.0%
$SC\text{-}Block_{Flooding}$	SupCon-Match	80	88%	53.9%	53.9%	53.9%	31	4.3%	74.3%	60.2%	66.5%	18	2.5%	70.5%	53.6%	60.9%
	Ditto	80	88%	98.0%	15.5%	26.7%	31	4.3%	92.3%	36.7%	52.3%	18	2.5%	93.0%	30.7%	46.2%
	DeepMatcher	80	88%	97.5%	16.4%	28.1%	31	4.3%	95.5%	31.1%	46.9%	18	2.5%	**95.0%**	27.9%	43.1%
SC-Block	SupCon-Match	20	2.2%	54.9%	53.0%	53.9%	16	2.2%	73.1%	56.9%	64.0%	35	4.9%	71.0%	48.7%	57.8%
	Ditto	20	2.2%	95.9%	20.5%	33.8%	16	2.2%	90.1%	38.8%	54.3%	35	4.9%	91.6%	25.8%	40.2%
	DeepMatcher	20	2.2%	96.9%	18.1%	30.4%	16	2.2%	94.5%	32.6%	48.4%	35	4.9%	94.4%	23.5%	37.6%
SRoBERTa	SupCon-Match	80	88%	3.1%	50%	5.9%	80	11.1%	55.8%	56.9%	56.3%	80	11.2%	49.9%	48.8%	49.3%
	Ditto	80	88%	5.2%	8.7%	6.5%	80	11.1%	70.7%	25.0%	36.9%	80	11.2%	63.3%	22.3%	33.0%
	DeepMatcher	80	88%	5.7%	6.4%	6.0%	80	11.1%	94.7%	27.5%	42.7%	80	11.2%	48.8%	26.8%	34.6%
Barlow Twins	SupCon-Match	80	88%	54.4%	54.4%	54.4%	80	11.1%	48.6%	55.8%	51.9%	80	11.2%	36.7%	52.0%	43.0%
	Ditto	80	88%	94.8%	12.6%	22.2%	80	11.1%	59.1%	25.9%	36.1%	80	11.2%	46.0%	23.9%	31.4%
	DeepMatcher	80	88%	95.2%	14.5%	25.1%	80	11.1%	95.1%	26.3%	39.5%	80	11.2%	44.7%	25.1%	32.9%

of the ability to retrieve matching pairs and the number of comparisons that need to be performed.

6 CONCLUSION

In this paper, we study the effectiveness of the state-of-the-art regularization method known as AdaFlood within the training procedure of SC-Block, a blocking method proposed by Brinkmann et al. [4]. Hereby, we develop an extended variant of SC-Block called SC-Block++, which uses AdaFlood to improve its overall effectiveness in finding candidate pairs. We evaluate the performance of SC-Block++ on its ability to construct candidate pairs, and compare its performance to that of the blocking schemes SC-Block, SC-Block with Flooding, SRoBERTa, and Barlow Twins on the data sets Walmart-Amazon, WDC_{20}, and WDC_{80}. Subsequently, we incorporate these blocking methods within state-of-the-art entity resolution pipelines and evaluate how the blocking methods influence the performance of these pipelines.

We find that SC-Block++ is an effective blocking scheme to construct candidate pairs as it outperforms the blocking methods SC-Block, SC-Block with Flooding, SRoBERTa, and Barlow Twins, in terms of the Pair Quality, Pair Completeness, and F_1^* score across all three data sets. These results are also reflected in the performance of the entity resolution pipelines when SC-Block++ is used during the blocking phase. When we tune the hyperparameter k in the nearest neighbour search of the blocking schemes, we find that SC-Block++ consistently achieves the lowest value of k, resulting in SC-Block++ creating less candidate pairs and therefore improving the efficiency of the entity resolution pipelines. This efficiency gain does not hamper the performance of the SC-Block++ pipelines, since these pipelines frequently outperform the other pipelines of the other blocking schemes in terms of precision, recall, and F_1.

There are several limitations to our research, which may lead to interesting directions for future research. To begin, in this work, we do not take into account the influence that the temperature parameter may have on the SupCon loss, which we use to train SC-Block. Wang and Liu [23] state that the temperature controls the trade-off between *uniformity* and *tolerance*. Uniformity measures how evenly the embeddings (constructed by the language model) are spread across the feature space, while tolerance represents how tolerant the SupCon loss is against samples that are semantically similar even though they are non-matching. Although uniformity is important for a model to learn separable factors between two non-matching samples [23], excessive pursuit to uniformity leads to the tolerance of a model worsening. Therefore, for future research, it may be interesting to study if a balance between uniformity and tolerance by tuning the temperature parameter improves the performance of SC-Block++, since, for example, it may be more resistant against corner cases.

Another limitation is that we do not consider that the size of the data sets may affect the performance of the blocking, as the data sets we consider here are all relatively small. A data set with more records and a larger vocabulary typically leads to more candidate pairs being constructed by a blocking method [4]. Hence, one could study if SC-Block++ is able to maintain a dominant performance over the other blocking schemes if the methods are applied to a larger data set.

Last, we borrow the learning rate and the number of epochs from Brinkmann et al. [4]. Yet, we may observe different results for the blocking methods if we tune these hyperparameters. Thus, one could study how tuning these parameters affects the results we obtain in this work.

A OPTIMAL FLOOD LEVEL FOR SC-BLOCK

We train $SC\text{-}Block_{Flooding}$ on the training splits of the three data sets we describe in Section 4. These models are trained for the flood values $\gamma \in \{0.000, 0.005, 0.010, \ldots, 0.025\}$. Subsequently, we utilize these variants of SC-Block to construct candidate pairs of the products in the validation splits of all three data sets. We perform the nearest neighbour search for $k \in \{1, 5, 10, 20, 40, 80\}$. In Table 4, we report the F_1^*

scores for each flood level across all values of k. Moreover, we compute the mean F_1^* scores to gain insight on the overall performance of the model across the values for k.

Although the optimal value of $\gamma = 0.000$ when we apply SC-Block$_{flooding}$ to Walmart-Amazon and WDC$_{20}$, we utilize $\gamma = 0.005$ for the other experiments, since SC-Block$_{flooding}$ is able to achieve the second-highest mean F_1^* scores for this value of γ (the variant of SC-Block with $\gamma = 0$ is included in the experiments any way as a baseline). Doing so allows us to compare the performance of SC-Block$_{flooding}$ to the other blocking methods.

Table 4: F_1^* scores of SC-Block$_{Flooding}$ for different values of the flood level across all three data sets

Data set	γ	F_1^*						Mean F_1^*
		$k=1$	$k=5$	$k=10$	$k=20$	$k=40$	$k=80$	
Walmart-Amazon	0.000	**58.1%**	**46.8%**	**38.7%**	**34.4%**	**30.5%**	**27.8%**	**39.4%**
	0.005	55.5%	44.8%	37.0%	31.7%	28.1%	25.9%	37.2%
	0.010	33.0%	31.9%	28.2%	25.6%	23.8%	22.4%	27.5%
	0.015	48.7%	40.5%	34.5%	29.3%	27.0%	24.5%	34.1%
	0.020	39.0%	34.6%	29.7%	26.8%	24.1%	22.5%	29.5%
	0.025	16.9%	20.0%	19.3%	17.9%	16.7%	15.5%	17.7%
WDC$_{20}$	0.000	29.7%	**82.0%**	**76.3%**	**70.9%**	**66.1%**	**61.5%**	**64.4%**
	0.005	31.7%	78.6%	74.0%	69.3%	64.7%	60.1%	63.1%
	0.010	31.6%	77.0%	72.1%	66.4%	62.9%	59.5%	61.6%
	0.015	**32.3%**	75.3%	72.1%	65.3%	60.9%	57.5%	60.6%
	0.020	30.3%	72.1%	71.1%	67.7%	61.6%	58.3%	60.2%
	0.025	29.7%	75.4%	72.5%	68.0%	62.8%	59.0%	61.2%
WDC$_{80}$	0.000	30.4%	69.5%	67.9%	62.5%	59.4%	57.6%	57.9%
	0.005	**32.5%**	**76.1%**	**70.6%**	**64.6%**	**61.5%**	57.9%	**60.5%**
	0.010	30.8%	67.4%	66.3%	64.3%	60.3%	**58.1%**	57.9%
	0.015	28.6%	70.9%	68.6%	62.6%	59.8%	57.3%	58.0%
	0.020	28.4%	67.3%	66.2%	61.5%	59.4%	56.8%	56.6%
	0.025	31.6%	68.7%	69.0%	63.2%	59.8%	57.6%	58.3%

REFERENCES

[1] Akiko Aizawa and Keizo Oyama. 2005. A fast linkage detection scheme for multi-source information integration. In *2005 International Workshop on Challenges in Web Information Retrieval and Integration (WIRI 2005)*. IEEE Computer Society, 30–39.

[2] Wonho Bae, Yi Ren, Mohamad Osama Ahmed, Frederick Tung, Danica J Sutherland, and Gabriel L Oliveira. 2023. AdaFlood: Adaptive Flood Regularization. *arXiv preprint arXiv:2311.02891* (2023).

[3] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 116, 32 (2019), 15849–15854.

[4] Alexander Brinkmann, Roee Shraga, and Christina Bizer. 2024. SC-Block: Supervised Contrastive Blocking Within Entity Resolution Pipelines. In *21st Extended Semantic Web Conference (ESWC 2024) (LCNS, Vol. 14664)*. Springer, 121–142.

[5] Peter Christen. 2012. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Transactions on Knowledge and Data Engineering* 24, 9 (2012), 1537–1555.

[6] Stephen Jose Hanson and Lorien Y. Pratt. 1988. Comparing Biases for Minimal Network Construction with Back-Propagation. In *1st International Conference on Neural Information Processing Systems (NIPS 1988)*. Morgan Kaufmann, 177–185.

[7] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[8] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. 2020. Do We Need Zero Training Loss After Achieving Zero Training Error?. In *37th International Conference on Machine Learning, (ICML 2020) (PMLR, Vol. 119)*. PMLR, 4604–4614.

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

[10] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *34th Conference on Neural Information Processing Systems (NIPS 2020)*. Curran Associates, 18661–18673.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

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

[13] 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 *2018 International Conference on Management of Data (SIGMOD 2018)*. ACM, 19–34.

[14] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. 2021. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment* 2021, 12 (2021), 124003.

[15] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2021. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys* 53, 2 (2021), 1–42.

[16] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2017. JedAI: The force behind entity resolution. In *The Semantic Web: (ESWC 2017) (LNCS, Vol. 10577)*. Springer, 161–166.

[17] Ralph Peeters and Christian Bizer. 2022. Supervised Contrastive Learning for Product Matching. In *The Web Conference 2022 (WWW 2022)*. ACM, 248–251.

[18] Ralph Peeters, Reng Chiz Der, and Christian Bizer. 2024. WDC Products: A multi-Dimensional Entity Matching Benchmark. In *27th International Conference on Extending Database Technology, (EDBT 2024)*. OpenProceedings, 22–33.

[19] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019)*. ACL, 3980–3990.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[21] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, and AnHai Doan. 2021. Deep learning for blocking in entity matching: a design space exploration. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2459–2472.

[22] Yingjie Tian and Yuqi Zhang. 2022. A comprehensive survey on regularization strategies in machine learning. *Information Fusion* 80 (2022), 146–166.

[23] Feng Wang and Huaping Liu. 2021. Understanding the Behaviour of Contrastive Loss. In *2021 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR 2021)*. Computer Vision Foundation/IEEE, 2495–2504.

[24] Yuexiang Xie, Zhen Wang, Yaliang Li, Ce Zhang, Jingren Zhou, and Bolin Ding. 2022. iFlood: A Stable and Effective Regularizer. In *10th International Conference on Learning Representations, (ICLR 2022)*. OpenReview.

[25] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive Approximation* 26, 2 (2007), 289–315.

[26] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *38th International Conference on Machine Learning, (ICML 2021) (PMLR, Vol. 139)*. PMLR, 12310–12320.

[27] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and Davd Page. 2020. Autoblock: A hands-off blocking framework for entity matching. In *13th International Conference on Web Search and Data Mining (WSDM 2020)*. ACM, 744–752.