Pre-trained Embeddings for Entity Resolution: An Experimental Analysis [Experiment, Analysis & Benchmark]

Alexandros Zeakis[1,2], George Papadakis[1], Dimitrios Skoutas[2], Manolis Koubarakis[1]
[1]National and Kapodistrian University of Athens, Greece {alzeakis,gpapadis,koubarak}@di.uoa.gr
[2]Athena Research Center, Greece {azeakis,dskoutas}@athenarc.gr

Abstract

Many recent works on Entity Resolution (ER) leverage Deep Learning techniques involving language models to improve effectiveness. This is applied to both main steps of ER, i.e., blocking and matching. Several pre-trained embeddings have been tested, with the most popular ones being fastText and variants of the BERT model. However, there is no detailed analysis of their pros and cons. To cover this gap, we perform a thorough experimental analysis of 12 popular language models over 17 established benchmark datasets. First, we assess their vectorization overhead for converting all input entities into dense embeddings vectors. Second, we investigate their blocking performance, performing a detailed scalability analysis, and comparing them with the state-of-the-art deep learning-based blocking method. Third, we conclude with their relative performance for both supervised and unsupervised matching. Our experimental results provide novel insights into the strengths and weaknesses of the main language models, facilitating researchers and practitioners to select the most suitable ones in practice.

PVLDB Reference Format:
Alexandros Zeakis[1,2], George Papadakis[1], Dimitrios Skoutas[2], Manolis Koubarakis[1]. Pre-trained Embeddings for Entity Resolution: An Experimental Analysis [Experiment, Analysis & Benchmark]. PVLDB, 15(9): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/alexZeakis/Embedings4ER.

1 Introduction

Entity Resolution (ER) is a crucial and challenging task for data integration [9], aiming to detect the different entity profiles that pertain to the same real-world object [8]. By deduplicating entity collections, ER facilitates a wide range of applications, from common data analytics tasks to advanced question answering [11].

Typically, ER solutions operate in two steps [6, 14]. First, *blocking* restricts the search space to the most likely matches, called *candidate pairs*, through a coarse-grained approach [39]. This is necessary in order to tame the inherently quadratic complexity of ER and allow it to scale to large volumes of data. Then, *matching* performs a fine-grained processing that examines each candidate pair to decide whether it constitutes a match [38].

Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Embedding text into numeric vectors has become a very common approach in NLP and related tasks [43]. Earlier models adopted sparse representations, based on bag-of-words or tf-idf [28]. However, these can only capture syntactic and not semantic similarity. This limitation is addressed by pre-trained embeddings models.

Motivated and inspired by this, the latest breakthroughs in ER leverage language models for both blocking [51] and matching [3, 31]. The following steps are typically involved in this process [12]. First, every given entity is transformed into a dense embedding vector. To perform blocking, the resulting vectors are indexed and a K-nearest neighbor query is issued for each entity to identify candidate pairs. These are then processed by a matching algorithm, which may operate in an unsupervised or a supervised mode. In the former case, the similarity between the embeddings vectors is computed and used as edge weights in a bipartite graph, where the nodes correspond to entities and the edges connect the candidate pairs. The graph is then split into disjoint sets of nodes, such that each of them contains all entities describing the same real-world object. In supervised matching, the embeddings vectors of the candidate pairs are fed as input to a binary classifier, typically a deep neural network, that decides whether each of them is a duplicate.

Over the years, numerous language models have been used in both ER steps. Few of them have been used for blocking: GloVe [12] and FastText [51, 60]. A much larger variety has been employed in matching: GloVe [12, 31], FastText [13, 22, 31, 32, 55, 58, 59] as well as BERT [41] and its main variants, i.e., XLNet, DistilBERT, RoBERTa and AlBERT [3, 4, 23, 35]. However, there is no systematic analysis of their relative performance in these ER tasks. We cover this gap, answering the following research questions:

(Q1) How large is their vectorization overhead?
(Q2) What is their relative performance in blocking?
(Q3) What is their relative performance in matching? How is it affected by fine-tuning in supervised matching?
(Q4) Is it possible to build high performing end-to-end ER pipelines based exclusively on pre-trained language models without providing them with labelled instances?

To answer these questions, we perform a thorough experimental analysis, involving 12 established language models: Word2Vec [29, 30] , GloVe [42], FastText [2], BERT [10], AlBERT [21], RoBERTa [26], XLNet [57], DistilBERT [47] as well as S-MPNet [48], S-T5 [44], S-DistilRoBERTa and S-MINILM [54]. Some of them are applied to ER for the first time. We apply them to 10 established real-world and 7 synthetic datasets, making the following contributions:

- We organize them into a taxonomy that facilitates the understanding of their relative performance and we discuss their core aspects like their dimensionality and context awareness.
- We examine the vectorization cost per model on GPU.
- We assess their blocking effectiveness, efficiency and scalability.

- We compare their relative performance for both supervised and unsupervised matching.
- We demonstrate that very high performance can be achieved in many datasets by an end-to-end solution that leverages the best language model both for blocking and matching without requiring any human intervention.

2 Related Work

Blocking. The first approach to leverage embeddings vectors for blocking is DeepER [12], which uses GloVe for the vectorization of entities and hyperplane LSH for indexing and querying. AutoBlock [60] goes beyond DeepER by leveraging FastText embeddings in combination with cross-polytope LSH. Yet, it treats blocking as classification task, requiring a large number of labelled instances to train its bidirectional-LSTM. DeepBlocker [51] is a generic framework for synthesizing deep learning-based blocking methods that support any language model. Most of the examined approaches, including the top-performing Auto-Encoder, leverage fastText embeddings, but support Word2Vec and GloVe too. It also considers two transformer-based models, which leverage Byte Pair Encoding, achieving slightly higher recall at the cost of much higher run-times. Overall, DeepBlocker with Auto-Encoder and fastText embeddings is the state of the art among these methods.

Matching. Here, language models are typically used by supervised deep learning-based techniques. The first one is again DeepER [12], which leverages GloVe. Its generalization, DeepMatcher [31], is a framework for deep learning-based matching algorithms that supports the main pre-trained static models, i.e., Word2Vec, GloVe and FastText, with the last one being the predefined option. Similarly, Seq2SeqMatcher [32], GraphER [22], CorDEL [55], MCAN [59], HierMatcher [13] and HIF-KAT [58] constitute individual approaches that also leverage FastText embeddings. The reason is its character-level operation, which addresses noise in the form of misspellings, while also supporting unseen terms.

More recent works focus on BERT-based models, due to their dynamic nature, which considers the context of terms to achieve higher effectiveness. The first such work is EMTransformer [3], which lays the ground for the analysis in Section 7.2. Originally, it considered BERT and its three main variants: XLNet, RoBERTa and DistilBERT. The same models are used by GNEM [4], which extends EMTransformer through a graph that captures the relations between all candidate pairs that are given as input to matching. GNEM also applies this idea to DeepMatcher, in combination with FastText embeddings. DITTO [23] extends EMTransformer by combining the BERT-based language models with external, domain-specific information (e.g., POS tagging) and data augmentation, which provides a higher number of artificial training instances.

JointBERT [41] goes beyond the classic binary classification definition of matching, by also supporting multi-class classification. The problem of automatically tuning the configuration parameters of deep learning-based matching algorithms is examined in [35], considering all BERT-based models in Section 3.2.

Gaps. We observe that none of these works considers the main SentenceBERT models (cf. Section 3.3). Moreover, no work has examined the performance of BERT models (cf. Section 3.2) on blocking. To the best of our knowledge, no work investigates the relative ER performance of the main language models in a systematic way. Closest to our work is an in depth analysis of how BERT works in the context of matching [34], but it disregards all other models as well as the task of blocking. Surveys [25], books [43] and tutorials [52] about language models are too generic, without any emphasis on ER, and thus, are orthogonal to our work. Our goal in this paper is to cover these important gaps in the literature.

Model	Dim.	$\|Seq.\|$	Param.	Blocking	Matching
Word2Vec (WC)	300	-	-	[51]	[31]
FastText (FT)	300	-	-	[60], [51]	[31], [32], [22], [55], [59], [13], [58]
GloVe (GE)	300	-	-	[12], [51]	[12], [31]
BERT (BT)	768	100	110M	-	[3], [4], [41], [23], [35]
AlBERT (AT)	768	100	12M	-	[35]
RoBERTa (RA)	768	100	125M	-	[3], [4] , [23], [35]
DistilBERT (DT)	768	100	66M	-	[3], [4], [23], [35]
XLNet (XT)	768	100	110M	-	[3], [4], [23], [35]
S-MPNet (ST)	768	384	110M	-	-
S-T5 (S5)	768	512	220M	-	-
S-DistilRoBERTa (SA)	768	512	-	-	-
S-MiniLM (SM)	384	256	33M	-	-

Table 1: The language models used in the evaluation.

3 Language Models Used in the Evaluation

We have used three categories of language models in our evaluation: (1) *Static models*, which associate every token with a fixed embedding vector; (2) *BERT-based models*, which vectorize every token based on its context; (3) *Sentence-BERT models*, which associate every sequence of tokens with a context-aware embedding vector.

For each category, we selected a representative set of language models based on the following criteria: (i) popularity in the ER and NLP literature, (ii) support for the English language, and (iii) availability of open-source implementation. Ideally, this implementation should include a documentation that facilitates the use of each model and all language models should be implemented in the same language so as to facilitate run-time comparisons. Thus, our experimental analysis relies on out-of-the-box, open-source implementations of the main language models that can be easily used by any practitioner that is not necessarily an expert in the field.

All static pre-trained and BERT-based models that are mentioned in Section 2 satisfy these criteria and, thus, are included in our analysis. Given that none of these ER works considers an established SentenceBERT model, we selected the top four ones from the SBERT library[1], based on their scores and overall need of resources. The technical characteristics of the selected models are summarized in Table 1. For each model, we indicate the vector dimensionality, the maximum sequence length, the number of parameters, and ER works that have used it for blocking or matching. Below, we briefly describe the selected models per category in chronological order.

3.1 Static pre-trained models

These models were introduced to capture semantic similarity in text, encapsulating knowledge from large corpora. They replace the traditional high-dimensional sparse vectors by low-dimensional

[1] https://www.sbert.net/docs/pretrained_models.html

dense ones, which define a mathematical space, where semantically similar words tend to have low distance.

Word2Vec [29, 30] is a shallow two-layer neural network that receives a corpus as input and produces the corresponding vectors per word. It employs a local context window, as a continuous bag-of-word (order-agnostic) or a continuous skip-gram (order-aware). The latter can link words that behave similarly in a sentence, but fails to utilize the statistics of a corpus.

GloVe [42] combines matrix factorization, i.e., the global co-occurrence counts, with a local context window, i.e., word analogy. It is trained on large corpora, such as Wikipedia, to provide pre-trained vectors for general use. Since it operates on a global dictionary, it identifies words with a specific writing and fails to detect slight modifications.

FastText [2] conceives each word as a group of n-grams instead of a single string. It is trained to vectorize n-grams. It then represents each word as the sum of its underlying n-grams.

3.2 BERT-based models

In static models, each word has a single representation, which is restrictive, since words often have multiple meanings based on their context. Context-awareness was introduced by the transformer models [53], which are a natural evolution of Encoders-Decoders [50]. The latter have many advantages, such as handling larger areas of text around a given word. Nonetheless, they cannot encapsulate any significant relationships between words. This has been fixed with the introduction of Attention [1], which facilitates the communication between each encoder / decoder, by sharing all of the corresponding hidden states and not just the last one. An extra optimization, suggested by the Transformer model, is the Multi-Head attention, which led each encoder to run in parallel. Another useful approach is the use of Positional Encoding for each token, which addresses polysemy, i.e., the fact that the same word has different meanings in different sentences.

BERT [10], which stands for "Bidirectional Encoder Representations from Transformers", was the next major step in the evolution of the transformer models. Its main contribution is the use of multiple transformers – only the encoder part, since it is a language representation model – to pre-train vectors for general use. These vectors can be further fine-tuned by adding an output layer for a wide variety of tasks. BERT is trained on two tasks: masked language modeling (MLM) and next sentence prediction (NSP). The former is token-based, since it tries to predict a masked token based on the unmasked tokens of a sentence. The latter is sentence-based, since it receives a first sentence as input and tries to predict whether a second sentence can follow it.

AlBERT [21], which stands for "A little BERT", is a lighter version of BERT. Base BERT comprises 12 encoders and 110M parameters with 768 hidden and equal embedding layers (cf. Table 1). AlBERT trains only the first encoder and then shares all its weights with the rest of the encoders. It also reduces the embedding layer by factorization to 128 layers. These reduce the total number of parameters to 12M, significantly lowering the training time.

RoBERTa [26] stands for "Robustly Optimized BERT pre-training Approach". Compared to BERT: (1) it is trained with more data and more and bigger batches; (2) it removes the next sentence prediction objective; (3) it changes the masked tokens per epoch to make the model more robust. It typically outperforms the original BERT [26].

DistilBERT [47], which stands for "Distillation BERT", is a lighter version of BERT that uses distillation [15, 46]. A second version of the original model is built, where only half of the attention layers are used – every second layer is omitted – and a special loss function is used in the training that compares the teacher (original BERT) with the student (DistilBERT).

XLNet [57] tries to overcome a certain drawback of BERT: the fact that it cannot utilize the knowledge of a predicted masked token as input for a second masked token, thus making each prediction independent and possibly false. XLNet introduces a variation of the MLM task, called permutation language modeling (PLM). The goal of the new task is to permute the tokens of one sentence in all possible matters without using any masked tokens.

3.3 SentenceBERT models

BERT-based models are mostly built to support token-based tasks. Supervised tasks that need a sentence representation may utilize the special token [CLS], but in regression or unsupervised tasks this produces a computational overhead, as all pair-wise combinations need to be fed into the model. Using [CLS] or averaging the last output layer is often worse than GloVe embeddings [45]. SBERT [45] fixes this problem by suggesting a Siamese architecture, i.e. two identical models, with each one taking as input one of the sentences. This architecture produces the corresponding vectors and then evaluates the combination of the two vectors, based on the defined task, e.g., the cosine similarity of two sentences. Note that this architecture is orthogonal to the underlying BERT model.

S-MPNet extends MPNet [48], which overcomes the drawbacks of BERT and XLNet in the MLM and PLM tasks, respectively. For the former, it solves the dependency between masked tokens predictions by permuting the tokens in a sentence. For the latter, it utilizes position information to reduce position discrepancy.

S-T5 extends T5 [44], an encoder-decoder model that aims to unify all NLP tasks under a single model. Instead of introducing a new model, it uses existing techniques. The text-to-text transfer transformer (T5) is an encoder-decoder model, where each transformer has been structured in the same way as in BERT. The rationale is that while BERT is an encoder model, the encoder-decoder model produces good results too and can be used for other tasks that an encoder cannot perform (e.g., text generation). T5 is trained on a dataset called "Colossal Clean Crawled Corpus", containing hundreds of gigabytes of clean English text from the Web.

S-DistilRoBERTa applies distillation to the RoBERTa model to produce a student, lighter model. This model is coupled with the Siamese architecture to produce the final model.

S-MiniLM extends MiniLM [54], which distills BERT to produce a much lighter student. Unlike DistilBERT and other distillation strategies [16, 49], which are bound to the architecture of the teacher layers, it mimics only the self-attention modules, which are the most important ones in the architecture. Thus, it can define the number of layers in each transformer, reducing the total number of required parameters. The distillation occurs in the pre-trained model to avoid fine-tuning the teacher, which is computationally expensive.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
Dat_1	$Rest_1$	Abt	Amz	DBLP	IMDb	IMDb	TMDb	Wmt	DBLP	IMDb
Dat_2	$Rest_2$	Buy	GPr.	ACM	TMDb	TVDB	TVDB	Amz	Scholar	DBP
$\|V_1\|$	339	1,076	1,354	2,616	5,118	5,118	6,056	2,554	2,516	27,615
$\|V_2\|$	2,256	1,076	3,039	2,294	6,056	7,810	7,810	22,074	61,353	23,182
$\|A_1\|$	7	3	4	4	13	13	30	6	4	4
$\|A_2\|$	7	3	4	4	30	9	9	6	4	7
$\|D\|$	89	1,076	1,104	2,224	1,968	1,072	1,095	853	2,308	22,863
$\|\bar{S}\|$	18.67	198.64	792.43	133.29	81,49	71.48	104.155	103.35	115.57	54.04

(a) Clean-Clean ER

	D_{s1}	D_{s2}	D_{s3}	D_{s4}	D_{s5}	D_{s6}	D_{s7}
	D_{10K}	D_{50K}	D_{100K}	D_{200K}	D_{300K}	D_1	D_{2M}
	10K	50K	100K	200K	300K	1M	2M
	12	12	12	12	12	12	12
	8,705	43,071	85,497	172,403	257,034	857,538	1,716,102
	84.32	84.21	84.34	84.30	84.30	84.31	84.32

(b) Dirty ER

Table 2: (a) The real datasets for Clean-Clean ER, and (b) the synthetic datasets for Dirty ER, in increasing total size, showing the number of entities ($|V_x|$), attributes ($|A_x|$), and duplicates ($|D|$), and the average sentence length in characters ($|\bar{S}|$).

	Dataset 1	Dataset 2	Total Pairs	Testing Pairs	Duplicates	Attributes
DSM_1	Abt	Buy	9,575	1,917	1,028	3
DSM_2	iTunes	Amazon	539	110	132	8
DSM_3	DBLP	ACM	12,363	2,474	2,220	4
DSM_4	DBLP	Scholar	28,707	5,743	5,347	4
DSM_5	Walmart	Amazon	10,242	2,050	962	5

Table 3: The datasets used in the Supervised Matching task.

4 Experimental Setup

4.1 Datasets

Main Datasets. Most experiments were conducted using the following ten real-world, established datasets for ER: D_1, which is offerred by OAEI 2010[2], contains descriptions of restaurants. D_2 contains products extracted from two online retailers, Abt.com and Buy.com [19]. D_3 comes from the same domain, matching products from Amazon.com and the Google Base data API (Google Pr.)[19]. D_4 involves bibliographic data from two publication repositories, DBLP and the ACM digital library [19]. D_5, D_6 and D_7 consist of three individual data sources, which comprise movie descriptions from imdb.com (IMDb) and themoviedb.org (TMDb) as well as TV shows from TheTVDB.com (TVDB) [33]. D_8 is another dataset from the product matching domain, involving descriptions from Walmart and Amazon [31]. Similar to D_4, D_9 contains bibliographic data from DBLP and Google Scholar [19]. Finally, D_{10} matches movies from IMDb and DBpedia [37], but has no overlap with the IMDb data source of D_5 and D_6.

All these data sets are publicly available through Zenodo[3] in CSV format. Their detailed characteristics are shown in Table 2(a). Note that all of them correspond to the *Clean-Clean ER* task, also known as *Record Linkage*, where the input comprises two individually duplicate-free, but possibly overlapping data sources and the goal is to detect the matching entities they share [9, 38].

Datasets for Scalability of Blocking. In addition, to evaluate blocking scalability, we employ the seven datasets shown in Table 2(b), which are widely used in the literature for this purpose [7, 18, 40]. They were artificially generated by Febrl [5], in the following way: clean entities were initially created by extracting real names (given and surname) and addresses (street number, name, postcode, suburb, and state names) from frequency tables of real census data. Next, duplicate entities were randomly generated according to realistic error rates and types (e.g., by inserting, deleting or replacing characters or words). In the end result, 40% of all entities are matching with at least another one. There are at most 9 duplicates per record and up to 3 and 10 modifications per attribute value and record, respectively.

Datasets for Supervised Matching. Finally, for supervised matching, we used the same five datasets as in [3], which are widely used in the literature [23, 31]. Their technical characteristics are reported in Table 3; 60% of all pairs form the training set, while the rest are equally split between the validation and test set. Note that most of them stem from the datasets in Table 2(a): DSM_1 is part of D_2, DSM_3 of D_4, DSM_4 of D_9 and DSM_5 of D_8. The only exception is DSM_2, which is not included in Table 2(a), due to the lack of its complete groundtruth. Note also that DSM_2-DSM_5 are *dirty datasets* in the sense that they include artificial noise in the form of missing and misplaced attribute values [23]. The goal of these settings is to increase their difficulty.

4.2 Settings

In all experiments, we consider all attribute values per entity, i.e., each entity is represented by the concatenation of all its attribute values. This setting, which is called *schema-agnostic*, inherently address misplaced attribute values (e.g., cases where person names are associated with their profession) and has shown to be effective for yielding high recall both in blocking [36, 40] and in matching [3]. Schema-based experiments, which exclusively consider the values of one or two specific attributes per dataset that combine high coverage with high distinctiveness, are reported in the extended version of our paper.[4]

All our code and datasets used in this experimental analysis are publicly available[5]. For the language models, we used the implementations provided by two highly popular Python packages: Gensim[6] and Hugging Face[7]. The former offers Word2Vec and FastText, while the latter provides all other models. All experiments were executed on a server with AMD Ryzen Threadripper 3960X 24-Core processor, 256 GB RAM and an RTX 2080Ti GPU, running Ubuntu 20.04. All experiments use the GPU, when possible.

5 Performance Comparison on Vectorization

Vectorization is the task of converting every textual entity in a given dataset into its embedding vector. In the schema-agnostic setting we consider, we represent each entity by a "sentence" that

[2] http://oaei.ontologymatching.org/2010/im
[3] https://zenodo.org/record/6950980
[4] https://github.com/alexZeakis/Embeddings4ER/tree/main/extended
[5] https://github.com/alexZeakis/Embeddings4ER
[6] https://radimrehurek.com/gensim
[7] https://huggingface.co

	WC	FT	GE	BT	AT	RA	DT	XT	ST	S5	SA	SM
init	31.2	155.7	48.18	4.5	4.0	5.1	4.2	5.8	8.6	9.6	9.0	8.2
D_1	0.1	0.2	0.2	2.2	2.3	2.1	1.3	3.5	1.1	2.9	0.7	0.8
D_2	0.1	1.5	0.2	2.9	2.4	2.3	2.1	3.2	3.4	10.6	1.8	1.4
D_3	0.9	9.6	0.4	10.1	6.5	6.2	8.6	7.8	10.2	38.4	5.7	3.3
D_4	0.2	2.4	0.3	5.7	5.0	5.0	3.9	7.1	5.0	16.4	2.7	2.2
D_5	0.4	3.8	0.4	13.0	12.2	12.0	8.5	18.0	10.6	36.1	6.0	4.5
D_6	0.6	5.5	0.5	15.0	13.5	13.4	10.1	19.7	15.0	52.2	8.1	5.4
D_7	0.4	3.6	0.4	11.5	10.6	10.5	7.5	15.7	9.7	31.3	5.4	4.1
D_8	1.0	9.7	0.8	27.3	24.3	23.6	18.5	35.4	28.1	83.3	14.5	11.0
D_9	2.4	26.9	1.9	71.0	63.0	62.7	47.9	92.9	57.1	187.4	30.8	24.4
D_{10}	1.0	10.5	1.2	47.8	46.5	45.0	29.8	70.9	28.2	89.0	16.0	15.2

Table 4: Vectorization time in seconds per model and dataset.

is formed by concatenating all its textual attributes. For Word2Vec and GloVe, which only support word embeddings, we tokenize this sentence into words and average their vectors to obtain a single one. FastText internally splits the sentence into smaller n-grams and aggregates their embeddings into a single vector. The rest of the models generate an embedding for the entire sentence.

Initialization. The initialization time of each model is shown in the first line of Table 4. It refers to the time taken to load the necessary data structures in main memory (e.g., a dictionary for the static models and a learned neural network for the dynamic ones), and is independent of the dataset used. The static models are inefficient due to the hash table they need to load into main memory to map tokens to embedding vectors. Regarding the dynamic models, we observe that the BERT-based ones are much faster than the SentenceBERT ones, as their average run-time is 4.7±0.8 and 8.9±0.6 seconds, respectively. This is due to the larger and more complex neural models that are used by the latter.

Transformation. The rest of Table 4 shows the total time required by each model to convert the entities of each dataset. Word2Vec and Glove are the fastest models by far, exhibiting the lowest processing run-times in practically all cases. Word2Vec is an order of magnitude faster than the third most efficient model per dataset, which interchangeably corresponds to FastText and S-MiniLM. Except for D_1, GloVe outperforms these two models by at least 6 times. In absolute terms, both Word2Vec and GloVe process the eight smallest datasets, D_1-D_8, in much less than 1 second, while requiring less than 2.5 seconds for the two larger ones.

Among the BERT-based models, DistilBERT is significantly faster than BERT, as expected, with its processing time being lower by 33%, on average. Note, though, that it is slower than FastText by >50%, on average, except for D_3. The next most efficient models of this category are ALBERT and RoBERTa, which outperform BERT by 11% and 13%, on average, respectively. XLNet is the most time consuming BERT-based model, being slower than BERT by ∼30%, on average across all datasets but D_3. This can be explained by the fact that D_3 has larger sentences, as shown in Table 2.

Regarding the SentenceBERT models, the lowest time is achieved by S-MiniLM, which, as mentioned above, is the third fastest model together with FastText. The second best model in this category is S-DistilRoBERTa, which is slower by 30%, on average. Both models are faster than BERT by 63% and 53%, respectively, on average. In contrast, the quite complex learned model of S-T5 yields the highest processing time among all models in all datasets but the smallest one. S-MPNet lies between these two extremes.

Summary. The static models excel in processing time, but suffer from very high initialization time. More notably, FastText is the slowest model in all datasets, but D_9, where S-T5 exhibits a much higher processing time. This means that FastText's high initialization cost does not pay off in datasets with few thousand entities like those in Table 2(a). On average, FastText requires 2 minutes per dataset. All other models vectorize all datasets in less than 1 minute, with very few exceptions: BERT over D_9, XLNet over D_9-D_{10} and S-T5 over D_8-D_{10}.

6 Performance Comparison on Blocking

Blocking receives as input the vectorized entities of the given datasets and returns as output a set of candidate pairs. To assess the performance of the language models on blocking, we again consider all datasets in Table 2(a).

6.1 Effectiveness

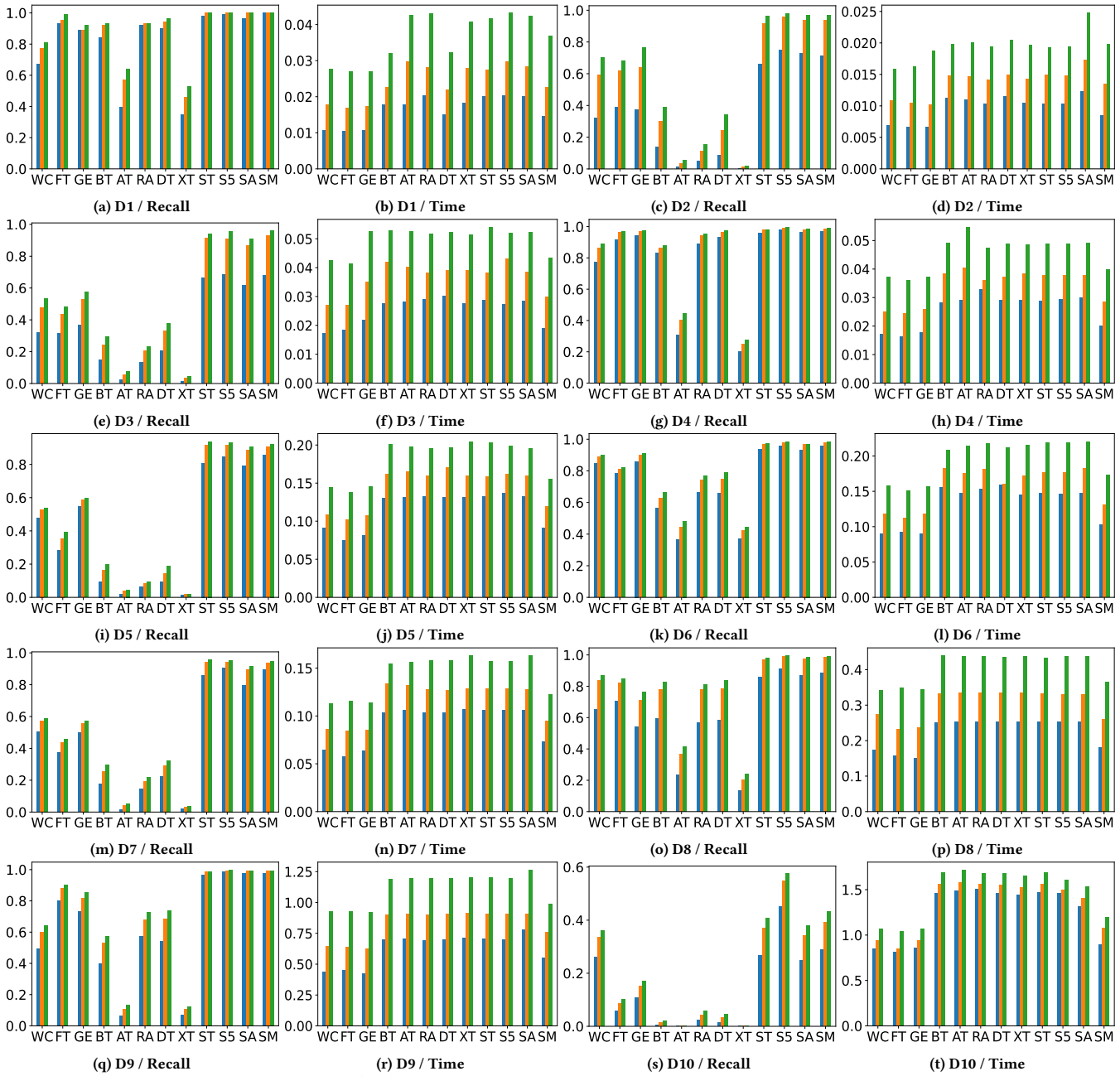
We measure the recall of the resulting candidate pairs, which is also known as pairs completeness [7, 18, 36]. Recall is the most critical evaluation measure for blocking, as it typically sets the upper bound for the subsequent matching step, i.e., a low blocking recall usually yields even lower matching recall, unless complex and time-consuming iterative algorithms are employed [6, 14]. In contrast, precision is typically low after blocking, due to the large number of false positives, but significantly raises after matching. We omit precision here, because all models have the same denominator (i.e., total number of candidates), due to the same k per query entity, thus recall and precision behave the same.

The experimental outcomes are shown in Figure 1. For all models, the recall increases with higher k across all datasets, which is expected, given that the set of candidate pairs gets larger. The same applies to the query time, due to the larger number of neighbors that are retrieved. More specifically:

• The SentenceBERT models consistently achieve the highest recall among all considered models. Among them, S-DistilRoBERTa is the least effective, followed in close distance by SMPNet, while ST-5 takes the lead in all datasets, leaving S-MiniLM in the second place. This applies to all values of k. The average recall of S-DistilRoBERTa, SMPNet, S-MiniLM and ST-5 is quite high in all datasets, except D_{10}, which abounds in noisy and missing values. In D_{10}, all SentenceBERT models actually exhibit a recall ∼0.55 for $k = 10$, which is the highest among all models. Excluding D_{10}, the average recall of all SentenceBERT models fluctuates between 0.960 (S-DistilRoBERTa) and 0.997 (ST-5). Their recall is also quite robust, with a standard deviation lower than 0.04 for all models.

• On the other extreme lie AlBERT and XLNet, which consistently underperform all other models. Their average recall across the datasets D_1-D_9 for k=10 is just 0.261 and 0.193, respectively. The situation is worse over D_{10}, where their recall is consistently lower than 0.05.

• The second best group of models are the static ones. They exhibit large differences between them for k=1 and k=5, but converge to quite similar recall for k=10. In all cases, though, GloVe is the top performer, with FastText and Word2Vec in the second and third place, respectively. In D_{10}, though, their relative performance is



(a) D1 / Recall (b) D1 / Time (c) D2 / Recall (d) D2 / Time
(e) D3 / Recall (f) D3 / Time (g) D4 / Recall (h) D4 / Time
(i) D5 / Recall (j) D5 / Time (k) D6 / Recall (l) D6 / Time
(m) D7 / Recall (n) D7 / Time (o) D8 / Recall (p) D8 / Time
(q) D9 / Recall (r) D9 / Time (s) D10 / Recall (t) D10 / Time

Figure 1: Recall and Blocking Time (sec) per method from real data per case. There are three values for $k \in \{1, 5, 10\}$.

quite different: FastText does not exceed 0.10, GloVe remains lower than 0.17, while Word2Vec ranges from 0.26 (k=1) to 0.36 (k=10).

• The remaining BERT-based models underperform all static pre-trained ones in almost all cases. Among them, DistilBERT is the top performer. BERT lies in the second place, but its advantage over RoBERTa is not significant. Their difference in terms of average recall is less than 2% in all cases. In the case of D_{10}, all these models achieve rather poor recall, which is lower than 0.035 even for k=10.

Summary. Figure 2 reports the ranking of each model per dataset with respect to recall for k=10. Looking at the rightmost column, which indicates the average position per model, we observe that the first four places are occupied by the SentenceBERT models, with ST-5 ranking first in most datasets and second in the rest. It is interesting that none of these models falls below the fourth place. The next three ranking positions correspond to the static models, with Word2Vec having the highest average one. Yet, FastText is the most stable one, fluctuating between positions 5 and 7, unlike the

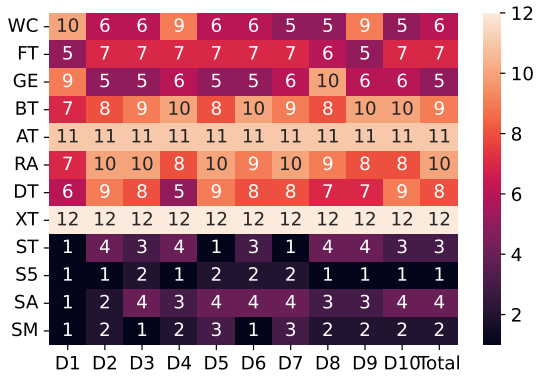


Figure 2: Method ranking wrt blocking recall (lower is better).

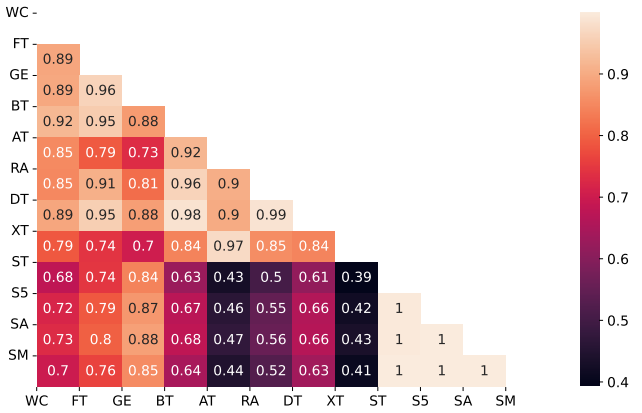


Figure 3: Pearson correlation of models wrt blocking recall.

other two models, which fall up to the 10^{th} place. Finally, BERT-based models are mapped to the last five positions. DistilBERT is the best one, ranked 8^{th} on average, while AlBERT and XLNet are constantly ranked 11^{th} and 12^{th}, respectively.

These patterns suggest a high correlation in terms of effectiveness between the models that belong to the same category. Indeed, the Pearson correlation with respect to recall for k=10 between the models of each category is quite high ($\geq$ 0.9), as shown in Figure 3. The correlation is equally high between the pre-trained static models and the SentenceBERT ones, due to the high performance of both categories. In contrast, the correlation between BERT-based and the other two categories is significantly lower: it fluctuates between 0.58 and 0.85 for the SentenceBERT models and between 0.69 and 0.86 for the static ones. In the latter case, DistilBERT is an exception, fluctuating between 0.84 and and 0.94, since it is the best-performing BERT model and, thus, it is closer to the static ones.

6.2 Efficiency

The execution time of blocking is very low for all models, not exceeding 0.5 seconds in most cases. The only exceptions are the two largest datasets, D_9 and D_{10}, which still require less than 2 seconds in all cases. The differences between the various models are rather insignificant. Yet, there are significant differences among the values of k we have considered. In most cases, the lower end in these ranges corresponds to the language models with the lowest dimensionality, namely the static ones (300) as well as S-MiniLM (385), and the higher end to the rest of the models, which involve 768-dimensional vectors (see Table 1). In more detail, fastText and S-T5 are consistently the fastest and slowest models, respectively. However, this overhead time is negligible in comparison to the vectorization time in Table 4. For this reason, the relative time efficiency remains the same as the one discussed in Section 5.

6.3 Comparison to state-of-the-art

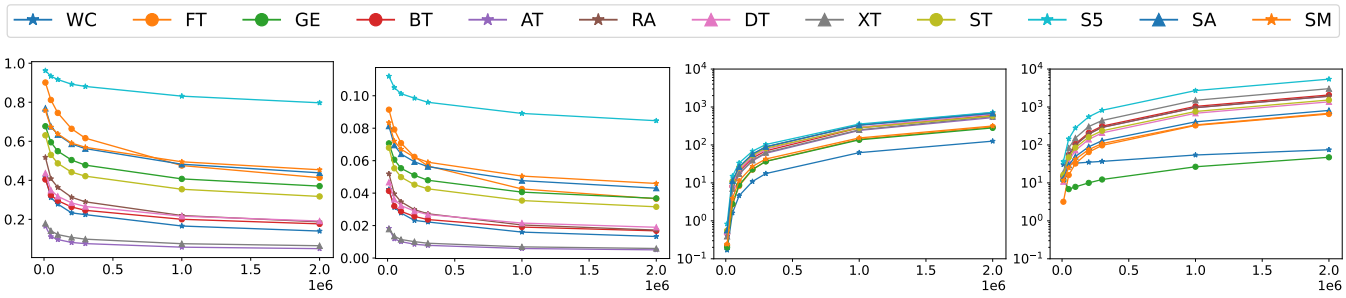
We now compare the best performing language model, S-T5, with the state-of-the-art blocking approach that is based on deep learning and embeddings vectors: DeepBlocker's Auto-Encoder with FastText embeddings. The relative performance of the two methods over all datasets in Table 2(a) appears in Table 5 (including vectorization time for S-T5). Note that since DeepBlocker is a stochastic approach, we take the average of 10 runs.

	DeepBlocker						S-T5					
	time (sec)			recall			time (sec)			recall		
	k=1	k=5	k=10	k=1	k=5	k=10	k=1	k=5	k=10	k=1	k=5	k=10
D1	8.0	8.0	17.4	0.96	1.00	1.00	12.5	12.5	12.5	0.99	1.00	1.00
D2	9.0	8.9	17.3	0.37	0.59	0.66	20.2	20.2	20.2	0.75	0.96	0.98
D3	18.9	18.9	35.6	0.36	0.56	0.62	48.0	48.0	48.1	0.69	0.91	0.96
D4	12.9	12.9	27.8	0.97	0.99	1.00	26.0	26.0	26.0	0.98	0.99	1.00
D5	24.3	24.5	62.5	0.59	0.77	0.80	45.8	45.9	45.9	0.85	0.91	0.93
D6	28.1	27.7	68.3	0.85	0.93	0.95	61.9	62.0	62.0	0.96	0.98	0.98
D7	21.8	21.7	49.4	0.78	0.84	0.85	41.0	41.0	41.1	0.91	0.94	0.96
D8	46.1	45.8	46.1	0.58	0.79	0.86	93.2	93.2	93.3	0.91	0.99	1.00
D9	110.6	111.1	270.4	0.82	0.90	0.91	197.7	197.9	198.2	0.99	1.00	1.00
D10	154.0	149.9	371.4	0.26	0.36	0.40	100.1	100.1	100.2	0.45	0.55	0.58

Table 5: Blocking performance of DeepBlocker vs S-T5.

Effectiveness. S-T5 consistently outperforms DeepBlocker's recall to a significant extent. The only exceptions are D_1 and D_4, where both methods achieve practically perfect recall – except for k = 1, where S-T5 is marginally better, by 3% and 1%, respectively. The reason is that D_1 involves a very low number of duplicate pairs in relation to the size of each data source, while D_4 contains relatively clean entities with long textual descriptions that are easy to match. In the remaining 8 datasets, the superiority of S-T5 is inversely proportional to the number of candidates per query entity, i.e., the lower k is, the larger is the advantage of S-T5.

Efficiency. DeepBlocker is consistently faster than S-T5 for k=1 and k=5 in all datasets but D_{10}. The reason for the slow operation of S-T5 is its high vectorization cost, which accounts for ~99% of the overall blocking time. This explains why its run-time is practically stable per dataset across all values of k. This is expected, though, given that S-T5 leverages 768-dimensional embeddings vectors, compared to 300-dimensional FastText vectors of DeepBlocker. Yet, for k=10, DeepBlocker is faster than S-T5 only in D_2 and D_3 (by 14.4% and 26%, respectively). The situation is reversed in D_1 and D_4-D_8, where S-T5 is faster by 14.1%, on average. The reason is that DeepBlocker does not scale well as the number of candidates per query entity increases, due to the high complexity of the deep neural network that lies at its core. Most importantly, DeepBlocker does scale well as the size of the input data increases: in D_{10}, S-T5 is faster by 1.5 times for $k \in \{1, 5\}$ and 3.7 times for k=10. This is further verified in Section 6.4.



(a) Recall (b) Precision (c) Blocking Time (sec-log) (d) Vectorization Time (sec-log)

Figure 4: Scalability analysis over the synthetic datasets in Table 2(b). The horizontal axis indicates the number of input entities.

Summary. Overall, we can conclude that S-T5 exhibits much higher blocking recall than DeepBlocker in all non-trivial cases at the cost of much higher overhead in case of small datasets and few candidates per query entity. In all other cases, S-T5 dominates DeepBlocker with respect to both effectiveness and time efficiency.

6.4 Scalability Analysis

Blocking must scale to large data volumes in order to restrict the input of matching to manageable levels, even in cases with millions of entities. Therefore, we need to assess how well the selected language models scale as the size of the input data increases.

To this end, we conduct an analysis using the seven datasets shown in Table 2(b). These correspond to *Dirty ER*, a.k.a., Deduplication, where a single data source containing duplicates is given as input [9, 38]. Here, due to the large dataset size, we use approximate instead of exact nearest neighbor search, applying FAISS [17] in combination with HNSW [27], which is a state-of-the-art algorithm for this task. Specifically, after vectorizing the input dataset as in Section 5, we index all entities using FAISS(HNSW); then, we query the resulting index with every entity e in order to retrieve the k=10 approximate nearest neighbors in terms of Euclidean distance – excluding e. The results appear in Figure 4.

Effectiveness. In Figure 4(a), we observe that recall consistently decreases for all models as we move from D_{10K} to D_{2M}. This is expected, given that the number of candidates increases quadratically with the size of the input data. The best performance clearly corresponds to S-T5: its recall over D_{2M} is quite satisfactory both in absolute terms (0.800) and in relation to the initial one over D_{10K} (0.962), as it drops by just 17%. The second best approach in most datasets is FastText, whose recall is reduced by 54%, from 0.901 to 0.415. On the other extreme lie AlBERT and XLNet: their recall is lower than 0.18 across all datasets (even for D_{10K}) and is reduced by 2/3 over D_{2M}. The rest of the models fluctuate between these two extremes and can be arranged into three groups according to their performance. The best group comprises S-DistilRoberta and S-MiniLM, which start from ~0.750 over the smallest dataset and end up ~40% lower at 0.450. The worst group includes Word2Vec, DistilBERT, BERT and RoBERTa, whose recall drops by 56%-66%, falling far below 0.2 over D_{2M}. GloVe and S-MPNet are located in the middle of the first two extremes: their recall is reduced by less than 50% and exceeds 0.32 over D_{2M}.

We observe almost the same patterns with respect to precision in Figure 4(b). This is due to the linear relation between the two measures, as explained earlier. Only minor variations occur in the context of Dirty ER, due to the different number of redundant candidate pairs, which are counted once (a candidate pair $<e_i, e_j>$ is redundant if e_j is included in the nearest neighbors of e_i and vice versa). Hence, there is a gradual decrease in precision for all models as the input size increases. This is larger than the decrease in recall by 2-3% in most cases, because the denominator of recall (i.e., the number of existing duplicates) increases at a slower pace than the denominator of precision (i.e., number of distinct candidate pairs).

Efficiency. In Figure 4(c), we observe that the execution time scales superlinearly, but subquadratically in all cases: as the number of input increases by 200 times from D_{10K} to D_{2M}, the run-times increase by up to 1,435 times. With the exception of the two most efficient models, Word2Vec (742 times) and S-T5 (873 times), all other models fluctuate between 1,053 (SMPNet) and 1,435 (XLNet) times. This is mainly attributed to the fact that FAISS(HNSW) trades high indexing time for significantly lower querying time. During indexing, it requires complex graph-based operations that involve long paths. The larger the input size, the longer these paths get, increasing the cost of their traversal and processing. Hence, the indexing time dominates the total time.

Finally, Figure 4(d) reports the evolution of vectorization time, which increases sublinearly with the size of the input data. The increase fluctuates between 127 (DistilBERT) and 165 (XLNet) times for all BERT-based models, thus remaining far below the raise in the input size (i.e., 200 from D_{10K} to D_{2M}). A similar behavior is exhibited by S-T5, but the rest of the SentenceBERT models achieve even better scalability, as their increase is reduced to 59 (S-MiniLM), 94 (S-MPNet) and 62 (XLNet). All these models are outperformed by the two most efficient static ones, Word2Vec and GloVe, which raise their run-times by just 2.5 and 4 times, respectively. The former actually remains practically stable up to D_{300K}, because its vectorization time is dominated by its high initialization time across the five smallest datasets. On the other extreme lies FastText, which is the only model that scales linearly with the size of the input data (i.e., its vectorization time increases by 205 from D_{10K} to D_{2M}).

In absolute terms, GloVe is consistently the fastest model in all datasets, but D_{10K}, with Word2Vec ranking second from D_{200K} on. They vectorize D_{2M} within 0.8 and 1.3 minutes, respectively. FastText is the second fastest model for the three smallest datasets, but converges to the fastest dynamic model, S-MiniLM, for the larger ones. They both need ~11 minutes to process D_{2M}. On the other extreme lies S-T5, which consistently exhibits the slowest

vectorization, with XLNet being the second worst model across all datasets. For D_{2M}, they take 90.3 and 50.6 minutes, respectively. The rest of the models can be grouped in three pairs: the slowest one, which is faster only than XLNet, includes BERT and RoBERTa, which vectorize D_{2M} within 34.6 and 31.9 minutes, respectively. The middle pair involves DistiBERT and S-MPNet, with the former being faster than the latter by 16%, on average (~25 minutes over D_{2M}). The fastest pair comprises S-MiniLM and S-DistilRoBERTa, with the former outperforming the latter by 15%, on average. These observations verify the conclusions drawn from Section 5 about the relative vectorization time of the considered models.

Comparison to the state-of-the-art. We applied DeepBlocker's Auto-Encoder with FastText embeddings to the datasets in Table 2(b). However, it runs out of memory already for the third largest dataset, due to its quadratic space complexity (i.e., it tries to create a 100K×100K float array for D_{100K}). For D_{10K} it takes 39.4 sec and scores 0.749 in recall and 0.130 in precision, whereas in D_{50K} it takes 463.2 sec and scores 0.676 and 0.116 accordingly. As a result, DeepBlocker is ~5% slower than S-T5 in D_{50K}, while its recall is lower by ~22%. The superiority of S-T5 is wider in D_{100K}, where DeepBlocker is almost ~3 times slower while yielding a recall lower by ~28%. FastText also outperforms DeepBlocker in all respects: it is an order of magnitude faster, while its recall is higher by at least 15%. S-MiniLM and S-DistilRoBERTa exhibit a recall similar to DeepBlocker in both datasets, while being faster by ~3 times and an order of magnitude over D_{10K} and D_{50K}, respectively. All other models are much faster, but less effective than DeepBlocker.

Summary. S-T5 is the clear winner among all models, due to its robust recall, which is maintained at acceptable levels for k=10 despite raising the input size by 200 times. Its precision also remains at very high levels for blocking. On average, 1 out of 10 candidate pairs involves duplicate entities. On the downside, its overall runtime is the highest among all models. Yet, it processes the largest dataset within ~1.5 hours, while it scales sub-linearly, increasing by just 164 times from D_{10K} to D_{2M}, as it is dominated by the vectorization time. S-T5 also scales much better than DeepBlocker.

7 Performance Comparison on Matching

There are two methodologies for addressing matching [6, 14]:

- *Unsupervised Matching* is formalized as a clustering task, where the candidate pairs are organized into disjoint groups, such that each group corresponds to a different real-world object (i.e., all pairs within each group are matching).
- *Supervised Matching* is formalized as a binary classification task, where the goal is to categorize the candidate pairs of blocking into two classes: `match` and `non-match`. Typically, a training set is used to learn a classification model, a validation set is used to choose its optimal configuration, and finally, a testing set is used to assess its performance on new, unknown instances.

7.1 Unsupervised Matching

We now examine the performance of all language models in the task of bipartite graph matching, i.e., in every dataset, each entity from the one data source is matched with at most one entity from the other data source. To this end, we apply Unique Mapping Clustering [20]. We calculate all similarities between all pairs of entities and iterate over them in descending order of similarity score, until all entities from the smallest data source are matched or there are no more pairs that exceed the specified similarity threshold. In this context, the current pair is marked as duplicates if none of its entities has already been matched with another entity. This approach has an increasing trend for recall, but a decreasing trend in precision. To maximize the effectiveness of each language model on every dataset, we consider all similarity thresholds in $[0, 1]$ with a step of 0.05 and select as optimal the one yielding the highest F1.

Effectiveness. Figure 5 reports the performance of all models with respect to recall, precision and f-measure (F1).

The SentenceBERT models consistently dominate all others. The best one is S-T5, achieving the highest F1 in seven out of the 10 datasets. In the remaining datasets, it is ranked second or third, lying very close to the top performer. On average, its distance from the maximum F1 is just 0.7%, being the lowest among all models. The worst case corresponds to D_6, where it lies 4.9% lower than the best method. The second best model is S-MiniLM, having the highest F1 in three datasets and the second lowest average distance from the maximum F1 (6.5%). The two next best models are S-MPNet and S-DistilRoBERTa, whose average distance from the top is 10.5%.

In absolute terms, their best performance corresponds to the bibliographic datasets, D_4 and D_9, where their F1 remains well over 0.9. The reason is that the long titles and list of authors facilitate the distinction between matching and non-matching entities. Also high (~0.8) is their F1 over D_2, which combines long textual descriptions with a 1-1 matching between the two data sources (i.e., every entity from the one source matches with one entity from the other). In all other datasets, their F1 ranges from 0.35 (D_{10}) to 0.77 (D_7), due to the high levels of noise they contain.

The second best group of models corresponds to the static ones. Based on the average distance from the top, GloVe is the best one (31.9%) followed by FastText (37.4%) and Word2Vec (39.4%). In absolute terms, their F1 remains rather low in all datasets except for the bibliographic ones: in D_4, it exceeds 0.9, lying very close to the SentenceBERT models, but in D_9, only FastText and Glove manage to surpass 0.7. The reason is that the entities from the Google Scholar are much more noisy and involve many more terminologies than D_4. In all other cases, their F1 falls (far) below 0.57.

Finally, the BERT-based models form the group of the least effective models. The worst performance is consistently exhibited by XLNet and AlBERT, as their F1 does not exceed 0.37 in any dataset. On average, their F1 is almost an order of magnitude (~87%) lower than the top one. In the other extreme lie DistilBERT and RoBERTa, with an average distance of ~55%. Finally, BERT fluctuates between these two extremes, with an average distance from the top equal to 62%. These three models score an acceptable F1 (~0.9) in the clean and easy D_4, but remain (far) below 0.54 in all other cases.

Figure 7 summarizes the ranking position of each model per dataset. The SentenceBERT models typically fluctuate between positions 1 and 4, the static ones between 5 and 7 and the BERT-based ones between 8 and 12. Moreover, the Pearson correlation of their F1 in Figure 6 shows an almost perfect dependency between the Sentence-BERT models and a very high one inside the group of static and BERT-based ones. The latter are weakly correlated with the SentenceBERT models. The static models have moderate correlation $(0.7 - 0.9)$ with the other two groups.

(a) D1 / Scores (b) D1 / Time (c) D2 / Scores (d) D2 / Time
(e) D3 / Scores (f) D3 / Time (g) D4 / Scores (h) D4 / Time
(i) D5 / Scores (j) D5 / Time (k) D6 / Scores (l) D6 / Time
(m) D7 / Scores (n) D7 / Time (o) D8 / Scores (p) D8 / Time
(q) D9 / Scores (r) D9 / Time (s) D10 / Scores (t) D10 / Time

Figure 5: Precision, Recall, F1 and Matching Time for Unsupervised Matching per dataset in Table 2(a).

Overall, we can conclulde that the BERT-based models require fine-tuning in order to be able to distinguish between the matching and the non-matching entities. The static models outperform them because they are independent of fine-tuning, but underperform the SentenceBERT ones, which are capable of enriching their embeddings with contextual positional evidence.

Efficiency. Time efficiency is assessed in terms of matching time (t_m), i.e., the time that is required for applying the Unique Mapping Clustering algorithm to all pairwise similarities using the optimal pruning threshold. Its matching time is typically much lower than half a second, except the largest dataset, D_{10}, where it raises to almost two seconds. This high efficiency in the context of a large number of candidate pairs should be attributed to its rather high similarity threshold, which fluctuates between 0.55 and 0.70, with a median (mean) of 0.625 (0.615), thus pruning the vast majority of pairs. Its lower effectiveness is combined with a much higher time efficiency. Even though its pruning threshold is lower by 0.05 or 0.10 than that of S-T5 – except for D_1, where it is higher by 0.05 — its matching time is significantly lower in most cases. In fact, it is faster than S-T5 by 3, 10 and 5 times over D_1, D_3 and D_{10},

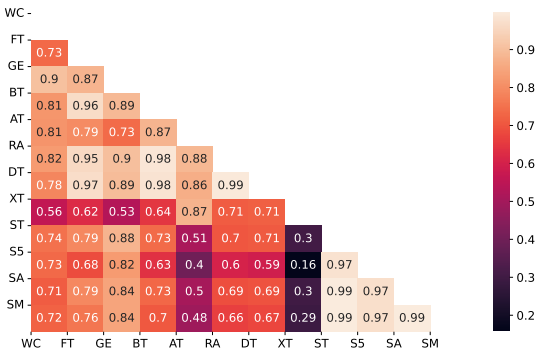


Figure 6: Pearson correlation with respect to F1 per pair of language models for unsupervised matching.

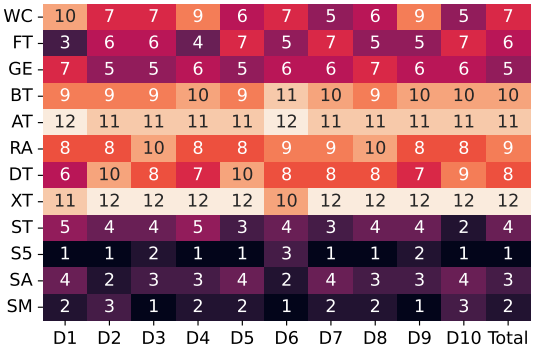


Figure 7: The ranking position of each method per dataset in unsupervised settings for Matching. Lower is better.

respectively. Their pruning thresholds are typically lower than S-T5, fluctuating between 0.50 and 0.65, thus yielding higher run-times. Yet, the matching time of S-DistilRoBERTa is consistently lower than a second, due to the distillation it conveys, while S-MPNet takes ~3 minutes to process D_{10}.

Comparison to the state-of-the-art. In this experiment we wanted to test an end-to-end framework of the best model so far, i.e. S-T5. To do so, we have used the matching algorithm described above, but instead of utilizing all pairs of entities, each query entity is allowed only k candidates, produced by Blocking. We have compared our framework with ZeroER [56]. As it can be seen in Table 6, ZeroER was not able to terminate in most datasets and in the rest cases, the blocking time was much larger than the total execution time of S-T5. In terms of effectiveness, again results are mostly in favor of S-T5, with the exception of D5, where S-T5 has a very low performance.

7.2 Supervised Matching

To examine the performance of all models in the task of supervised matching, we combine them with *EMTransformer* [3]. We selected this approach among the open-source deep learning-based matching algorithms, because it achieves state-of-the-art performance, while relying exclusively on the embedding models. This is in contrast to DITTO [23], which leverages external information, such as POS tagging. This is also in contrast to DeepMatcher [31], which is crafted for the static models, unlike EMTransformer,

	ZeroER					S-T5				
Dataset	BLK	MTCH	Prec.	Rec.	F1	VECT	MTCH	Rec.	Prec.	F1
D1	2.12	0.38	0.00	0.00	0.00	12.44	0.00	0.99	0.32	0.48
D2	1727.55	70.70	0.86	0.32	0.46	20.21	0.00	0.82	0.85	0.83
D3	-	-	-	-	-	47.92	0.00	0.64	0.52	0.57
D4	9595.28	113.19	0.97	0.97	0.97	25.97	0.01	0.98	0.97	0.97
D5	1291.44	9.77	0.91	0.95	0.93	45.73	0.01	0.74	0.20	0.31
D6	-	-	-	-	-	61.91	0.02	0.87	0.21	0.34
D7	1598.54	338.19	0.86	0.90	0.88	41.03	0.01	0.90	0.47	0.62
D8	-	-	-	-	-	92.97	0.01	0.69	0.24	0.35
D9	-	-	-	-	-	197.02	0.01	0.95	0.88	0.91
D10	-	-	-	-	-	98.66	0.09	0.49	0.55	0.52

Table 6: ZeroER vs S-T5 on Unsupervised Matching.

	DSM_1	DSM_2	DSM_3	DSM_4	DSM_5
BERT	0.843	0.926	0.988	0.949	0.819
AlBERT	0.833	0.894	0.989	0.948	**0.844**
RoBERTa	**0.866**	**0.962**	0.982	**0.956**	0.831
DistilBERT	0.817	0.792	0.991	0.947	0.720
XLNet	0.864	0.893	0.986	0.953	0.760
(a) BERT-based Models					
S-MPNet	0.862	**0.962**	0.989	0.952	0.824
S-DistilRoBERTa	0.861	0.830	0.980	**0.956**	0.790
S-MiniLM	0.857	0.857	**0.992**	0.946	0.739
(b) SentenceBERT Models					
DITTO	0.893	0.957	0.990	0.958	0.857
DeepMatcher+	0.628	0.794	0.981	0.938	0.538

Table 7: F1 score of models for Supervised Matching.

	DSM_1		DSM_2		DSM_3		DSM_4		DSM_5	
	t_t	t_e	t_t	t_e	t_t	t_e	t_t	t_e	t_t	t_e
BT	1,753.2	11.2	66.2	0.4	1,521.3	9.6	2,611.1	15.8	1,089.9	6.8
AT	1,687.5	12.0	62.8	0.5	1,443.8	10.3	2,417.5	17.0	1,023.1	7.3
RA	1,768.3	11.1	66.0	0.4	1,543.3	9.6	2,657.2	15.7	1,106.6	6.8
DT	891.3	5.5	33.9	0.2	777.9	4.8	1,339.9	7.9	558.4	3.4
XT	2,911.7	24.1	91.9	0.7	2,117.0	16.0	3,190.8	20.8	1,421.0	10.1
(a) BERT-based Models										
ST	1,633.8	10.5	61.5	0.4	1,404.4	8.9	2,399.8	14.4	1,008.9	6.3
SA	810.7	5.0	31.3	0.2	716.9	4.3	1,250.2	7.1	517.8	3.1
SM	724.2	4.2	25.4	0.2	579.9	3.3	996.9	5.3	416.3	2.3
(b) SentenceBERT Models										

Table 8: Time efficiency of the dynamic models in Supervised Matching over the datasets in Table 3. The training (t_t) and testing (t_e) times are measured in minutes and seconds, respectively, and correspond to the average after 5 runs.

which supports dynamic ones, too. In fact, the static language models are excluded from this analysis, because their representations cannot be adjusted after training. As a result, they are incompatible with EMTransformer. S-T5 is also excluded, because it follows the Encoder-Decoder architecture, which is different than the other models which only have the Encoder component.

The original implementation of EMTransformer disregards the validation set and evaluates each model directly on the testing set. However, this results in overfitting, as noted in [24]. We modified the code so that it follows the standard approach in the literature: for each trained model, the validation set is used to check whether it maximizes F1 and this model is then applied to the testing set [23, 31]. For this analysis, we used the five datasets shown in Table 3.

Effectiveness. Table 7 summarizes the results of our experiments with respect to effectiveness (F1 score) over the datasets in Table 3 (the best performance per dataset is highlighted in bold). As baseline methods we have included the performance of DITTO and DeepMatcher, as reported in [23]. We observe that RoBERTa is the most robust model in terms of effectiveness. It achieves the highest F1 in most datasets, ranking first on average. In fact, the mean distance from the top F1 across all datasets amounts to just 0.5%. It is followed in close distance by the second best model, S-MPNet, which is top performer in one dataset (DSM_3) and its average distance from the top F1 is 0.7%. The rest of the models are sorted in the following order according to their average distance from the top performance: BERT, AlBERT, XLNet, S-DistilRoBERTa, S-MiniLM.

Even the least effective model, though, is just 5% worse than the best one, on average. The reason for this is the strong correlation between all considered models: high effectiveness for one model on a specific dataset implies similarly high effectiveness for the rest of the models. This pattern should be partially attributed to the DSM_3 and DSM_4 datasets, where the difference between the maximum and the minimum F1 is less than 1.3%. These datasets convey relatively clean values, even though in both cases artificial noise has been inserted in the form of misplaced values. This means that the duplicate entities share multiple common tokens in the schema-agnostic settings we are considering. As a result, even classic machine learning classifiers that use string similarity measures as features achieve very high F1 (>0.9) in these datasets (see Magellan in [23, 31]). The rest of the datasets are more challenging, due to the terminologies they involve (e.g., product names). As a result, the difference between the maximum and minimum F1 of these models ranges from 4.9% (DSM_1) to 17% (DSM_2).

These experiments indicate that the BERT-based models match and even outwit the SentenceBERT ones, unlike for the blocking and the unsupervised matching tasks, due to the fine-tuning of their last layer.

Efficiency. We report the time efficiency of the language models with respect to training time (t_t) and testing time (t_r) in Table 8. We observe that S-MPNet constitutes the best choice for applications that emphasize time efficiency at the cost of slightly lower effectiveness: its training and prediction time are consistently lower than that of RoBERTa by 8.7% and 7.2%, respectively, on average. Compared to XLNet, which is consistently the slowest model in all respects among all datasets, RoBERTa excels in both effectiveness and run-time, as it is faster by $\geq$ 38%, on average. ALBERT achieves slightly lower training times (by 6.7%) than RoBERTa at the cost of higher prediction times (by 7.6%). The rest of the models reduce both run-times to a half, on average, at the cost of significantly lower F1 ($\ll$10%).

Comparison to the state-of-the-art. Comparing the language models to DeepMatcher+, which leverages FastText embeddings, we observe that they outperform it in practically all cases. This is particularly true in DSM_1 and DSM_5, where the worst language model (DistilBERT) exceeds DeepMatcher+ by ~20%. DITTO, on the other hand, is directly comparable to the best performing language model in each dataset. In DSM_2, DSM_3 and DSM_4, DITTO's F1 is lower by just $\leq$0.5%. In DSM_1 and DSM_5, though, DITTO outperforms all language models by 3% and 1.5%, respectively. This should be attributed to the external knowledge and the data augmentation. The effect of these features is more clear when comparing DITTO to the language model that lies at its core, i.e., RoBERTa. On average, across all datasets, the latter underperforms DITTO by 1.3%.

8 Discussion & Conclusions

We examined two types of downstream tasks: the unsupervised ones, which include blocking and bipartite graph matching, and supervised ones, which includes Supervised Matching. In the former tasks, the language models we considered can be clustered into three groups, according to their performance. The top performing group includes the SentenceBERT models, which are able to distinguish between matching and non-matching entities without any fine-tuning of their top attention layers. Among them, S-T5 excels in effectiveness, but is slower, while S-MiniLM offers a better balance, trading slightly lower effectiveness for significantly higher time efficiency. The second best group includes the static pretrained models, which are by far the most efficient ones. The worst group in terms of effectiveness includes the BERT-based models, which typically fail to distinguish between matching and non-matching entities. The reason is that they require fine-tuning, unlike the static models, which anyway cannot be fine-tuned. These patterns apply equally to the blocking and the bipartite graph matching tasks.

This is not true for the Supervised Matching task, which applies only to the dynamic language models. Yet, the difference between the BERT and the SentenceBERT models is minimized, due to fine-tuning. In fact, most BERT models outperform most SentenceBERT models in terms of F1. The latter are slightly faster both in terms of training and prediction time, because they are crafted for parsing entire sentences, i.e., they are tailored to the schema-agnostic settings we considered in our experiments. Note, though, that similar conclusions can be drawn from the corresponding schema-based settings, which are reported in the extended version of our work.

The static pre-trained models serve as baselines for word embeddings. We expect them to be fast in transforming text to vectors, but with low precision and recall in blocking and matching. The BERT-based models should provide better word embeddings due to their dynamic nature. They should also provide possibly better sentence embeddings as well, since the special [CLS] token captures the whole sentence. Moreover, BERT should be the baseline of this category. ALBERT and DistilBERT offer lighter, i.e., more time efficient, versions compared to BERT, while RoBERTa has a slight overhead, but better results, since it is a more robust model. Finally, XLNet deals with a slightly different problem than original BERT, but essentially builds same model and, thus, should show no actual difference in our tasks.

The SBERT models argue that the [CLS] token from BERT is insufficient and, thus, each model in this category should have better scores in terms of precision and recall than BERT models. S-DistilRoBERTa and S-MiniLM offer lighter models, while maintaining the same performance. The poor performance of BERT-based models in both blocking and matching is caused by the fact that without fine-tuning, they essentially assign very high similarity scores to all candidate pairs, thus failing to distinguish between matching and non-matching ones.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[3] Ursin Brunner and Kurt Stockinger. 2020. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *EDBT*. 463–473.

[4] Runjin Chen, Yanyan Shen, and Dongxiang Zhang. 2020. GNEM: A Generic One-to-Set Neural Entity Matching Framework. In *WWW*. 1686–1694.

[5] Peter Christen. [n.d.]. Febrl -: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *SIGKDD*, Ying Li, Bing Liu, and Sunita Sarawagi (Eds.). 1065–1068.

[6] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer.

[7] Peter Christen. 2012. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. *IEEE Trans. Knowl. Data Eng.* 24, 9 (2012), 1537–1555.

[8] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42.

[9] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. *Entity Resolution in the Web of Data.* Morgan & Claypool Publishers.

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

[11] Xin Luna Dong and Divesh Srivastava. 2013. Big Data Integration. *Proc. VLDB Endow.* 6, 11 (2013), 1188–1189.

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

[13] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. 2020. Hierarchical Matching Network for Heterogeneous Entity Resolution. In *IJCAI*. 3665–3671.

[14] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity Resolution: Theory, Practice & Open Challenges. *Proc. VLDB Endow.* 5, 12 (2012), 2018–2019.

[15] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[16] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[18] Batya Kenig and Avigdor Gal. 2013. MFIBlocks: An effective blocking algorithm for entity resolution. *Inf. Syst.* 38, 6 (2013), 908–926.

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

[20] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. 2013. SIGMa: simple greedy matching for aligning large knowledge bases. In *KDD*. 572–580.

[21] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[22] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks. In *IAAI*. 8172–8179.

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

[24] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *CoRR* abs/2004.00584 (2020). https://arxiv.org/abs/2004.00584

[25] Qi Liu, Matt J. Kusner, and Phil Blunsom. 2020. A Survey on Contextual Embeddings. *CoRR* abs/2003.07278 (2020). https://arxiv.org/abs/2003.07278

[26] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[27] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[28] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval.* Cambridge University Press.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[31] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*. 19–34.

[32] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. Deep Sequence-to-Sequence Entity Matching for Heterogeneous Entity Resolution. In *International Conference on Information and Knowledge Management*. 629–638.

[33] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. 2021. EAGER: Embedding-Assisted Entity Resolution for Knowledge Graphs. *CoRR* abs/2101.06126 (2021).

[34] Matteo Paganelli, Francesco Del Buono, Andrea Baraldi, and Francesco Guerra. 2022. Analyzing How BERT Performs Entity Matching. *Proc. VLDB Endow.* 15, 8 (2022), 1726–1738.

[35] Matteo Paganelli, Francesco Del Buono, Pevarello Marco, Francesco Guerra, and Maurizio Vincini. 2021. Automated machine learning for entity matching tasks. In *EDBT*.

[36] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. 2015. Schema-agnostic vs Schema-based Configurations for Blocking Methods on Homogeneous Data. *Proc. VLDB Endow.* 9, 4 (2015), 312–323.

[37] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. 2011. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*. 535–544.

[38] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution.* Morgan & Claypool Publishers.

[39] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2021. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.* 53, 2 (2021), 31:1–31:42.

[40] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. *Proc. VLDB Endow.* 9, 9 (2016), 684–695.

[41] Ralph Peeters and Christian Bizer. 2021. Dual-Objective Fine-Tuning of BERT for Entity Matching. *Proc. VLDB Endow.* 14, 10 (2021), 1913–1921.

[42] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[43] Mohammad Taher Pilehvar and José Camacho-Collados. 2020. *Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning.* Morgan & Claypool Publishers.

[44] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[45] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019).

[46] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

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

[48] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems* 33 (2020), 16857–16867.

[49] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2019. Mobilebert: Task-agnostic compression of bert by progressive knowledge transfer. (2019).

[50] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[51] 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. *Proc. VLDB Endow.* 14, 11 (2021), 2459–2472.

[52] Immanuel Trummer. 2022. From BERT to GPT-3 Codex: Harnessing the Potential of Very Large Language Models for Data Management. *Proc. VLDB Endow.* 15, 12 (2022), 3770–3773.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[54] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems* 33 (2020), 5776–5788.

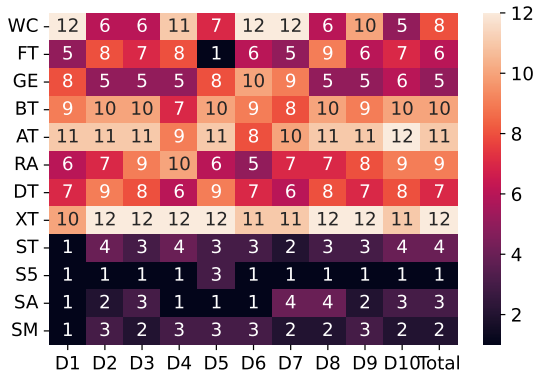


Figure 8: Method ranking wrt blocking recall (lower is better) (Schema-Based).

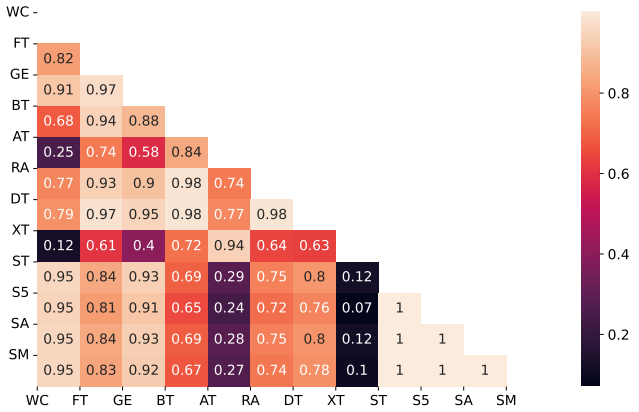


Figure 9: Pearson correlation of models wrt blocking recall (Schema-Based).

[55] Zhengyang Wang, Bunyamin Sisman, Hao Wei, Xin Luna Dong, and Shuiwang Ji. 2020. CorDEL: A Contrastive Deep Learning Approach for Entity Linkage. In *ICDM*. 1322–1327.

[56] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1149–1164.

[57] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[58] Zijun Yao, Chengjiang Li, Tiansi Dong, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Yichi Zhang, and Zelin Dai. 2021. Interpretable and Low-Resource Entity Matching via Decoupling Feature Learning from Decision Making. In *ACL/IJCNLP*. 2770–2781.

[59] Dongxiang Zhang, Yuyang Nie, Sai Wu, Yanyan Shen, and Kian-Lee Tan. 2020. Multi-Context Attention for Entity Matching. In *WWW*. 2634–2640.

[60] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. 2020. AutoBlock: A Hands-off Blocking Framework for Entity Matching. In *WSDM*. 744–752.

9 Appendix I: Schema-based Experiments

In Figures 8, 9 and 12 we see results of Blocking on schema-based settings. The trend is similar to schema-agnostic, i.e. static models are faster, but not that efficient, while SBERT models are dominant. A similar trend is followed in Figures 13, 11 and 10 for Unsupervised Matching.

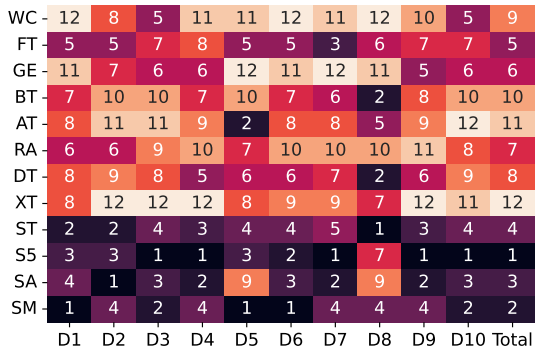


Figure 10: The ranking position of each method per dataset in unsupervised settings for Matching. Lower is better (Schema-Based).

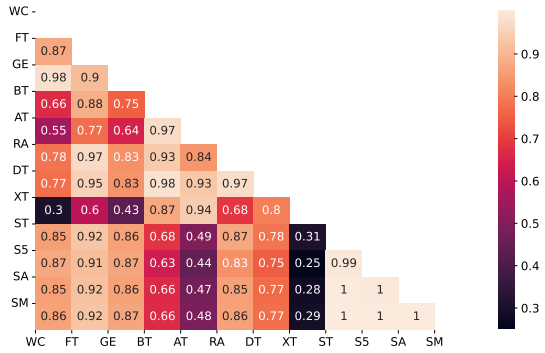
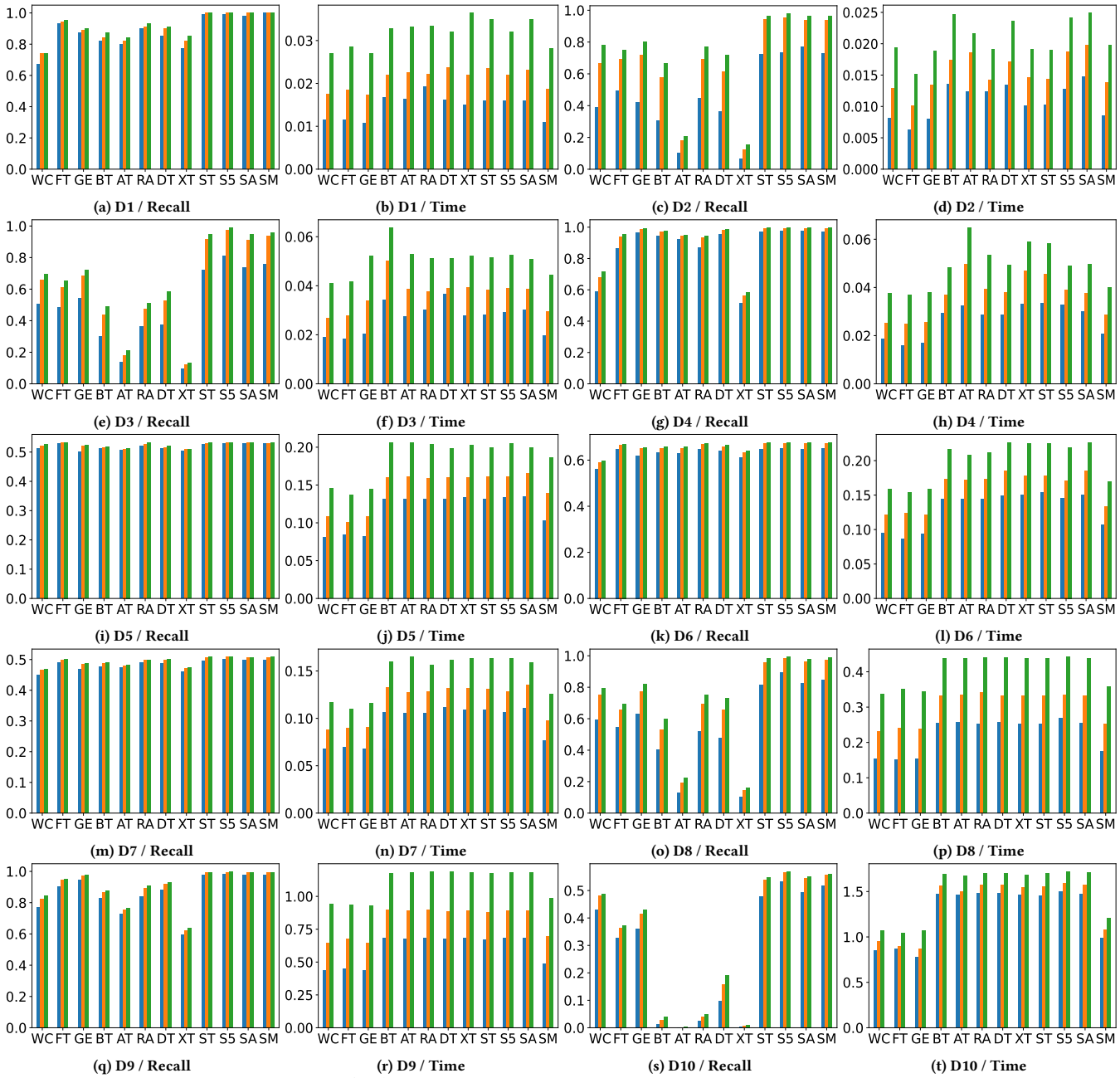


Figure 11: Pearson correlation with respect to F1 per pair of language models for unsupervised matching (Schema-Based).

10 Appendix II: Complementary plots for comparison

In this appendix we want to present an alternative approach to compare models in the tasks of Blocking and Unsupervised Matching. We examine the models under three groups, namely: static, BERT and SBERT, based in Table 1 and with the metric of Recall. It is clear from Figures 14 and 15, that SBERT models have the best and most coherent behaviour in all datasets and especially S-T5.



(a) D1 / Recall (b) D1 / Time (c) D2 / Recall (d) D2 / Time

(e) D3 / Recall (f) D3 / Time (g) D4 / Recall (h) D4 / Time

(i) D5 / Recall (j) D5 / Time (k) D6 / Recall (l) D6 / Time

(m) D7 / Recall (n) D7 / Time (o) D8 / Recall (p) D8 / Time

(q) D9 / Recall (r) D9 / Time (s) D10 / Recall (t) D10 / Time

Figure 12: Recall and Blocking Time (sec) per method from real data per case. There are three values for $k \in \{1, 5, 10\}$ (Schema-Based).

(a) D1 / Scores (b) D1 / Time (c) D2 / Scores (d) D2 / Time

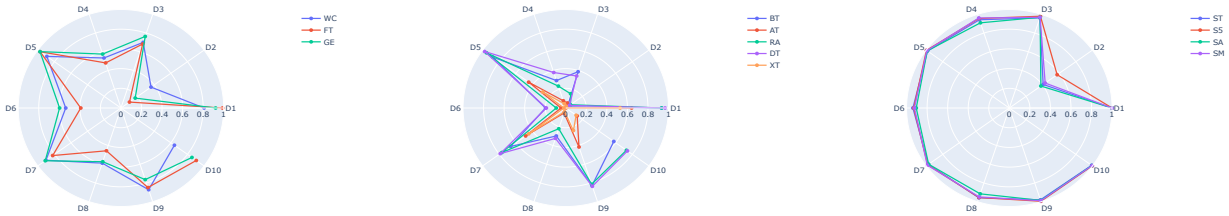
(e) D3 / Scores (f) D3 / Time (g) D4 / Scores (h) D4 / Time

(i) D5 / Scores (j) D5 / Time (k) D6 / Scores (l) D6 / Time

(m) D7 / Scores (n) D7 / Time (o) D8 / Scores (p) D8 / Time

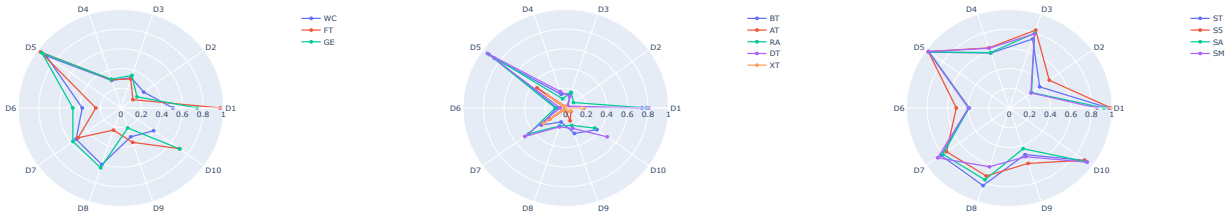
(q) D9 / Scores (r) D9 / Time (s) D10 / Scores (t) D10 / Time

Figure 13: Precision, Recall, F1 and Matching Time for Unsupervised Matching per dataset in Table 2(a) (Schema-Based).



(a) Static (b) BERT (c) SBERT

Figure 14: Recall comparison for each group of models in Blocking (Schema-agnostic).



(a) Static (b) BERT (c) SBERT

Figure 15: Recall comparison for each group of models in Unsupervised Matching (Schema-agnostic).