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 [10], aiming to detect the different entity profiles that pertain to the same real-world object [9]. By deduplicating entity collections, ER facilitates a wide range of applications, from common data analytics tasks to advanced question answering [12].

Typically, ER solutions operate in two steps [7, 15]. First, *blocking* restricts the search space to the most likely matches, called *candidate pairs*, through a coarse-grained approach [43]. 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 [42].

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 [47]. Earlier models adopted sparse representations, based on bag-of-words or tf-idf [30]. 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 [55] and matching [3, 33]. The following steps are typically involved in this process [13]. 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.

As an example, consider the two entity collections in Figure 1. They comprise data about smartwatches from shopping websites. We can see that e_1 matches with $_k$, e_2 with e_j and e_3 with e_k, despite their substantially different descriptions. To use the considered language models in ER, the first step is vectorization, which in the schema-agnostic settings, converts every entity profile into a single numeric dense vector of fixed dimensionality. Entity e_1 is converted to vector v_1, entity e_2 to vector v_2 and so on and so forth. A successful language model assigns semantically similar entities to vector of low distance, as in our example. Next, FAISS(HNSW) indexes the vectors of the first entity collections, converting them to a graph structure, called Hierarchical Navigable Small-World graph, that allows for efficient searches for nearest neighbors. The entities of the second entity collection are actually posed as queries to the FAISS(HNSW) index in the next step, called querying, which retrieves the k most similar vectors from the other entity collection per entity. The resulting pairs of similar vectors form the set of candidate pairs. For each pair of candidates, we estimate the (Euclidean) similarity of the corresponding vectors, associating it with a similarity score. The pairs with a scores higher than a threshold (0.5 in our example) are then processed by the Unique Mapping Clustering algorithm, which essentially associates every entity with its most similar candidate, disregarding all others. These matched entities form the output of the entire process.

Over the years, numerous language models have been used in both ER steps. Few of them have been used for blocking: GloVe [13] and FastText [55, 65]. A much larger variety has been employed

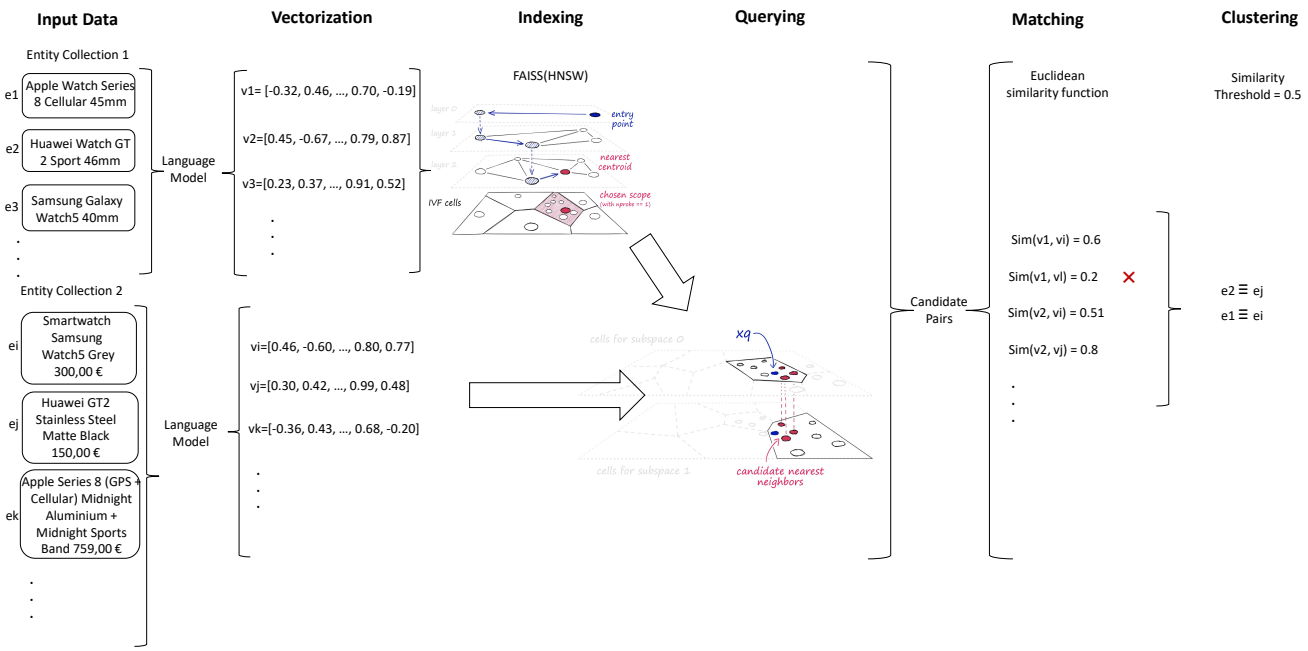


Figure 1: An example illustrating the end-to-end unsupervised approach to Entity Resolution, based on language models.

in matching: GloVe [13, 33], FastText [14, 23, 33, 35, 60, 63, 64] as well as BERT [45] and its main variants, i.e., XLNet, DistilBERT, RoBERTa and AlBERT [3, 5, 25, 38]. 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 [31, 32] , GloVe [46], FastText [2], BERT [11], AlBERT [22], RoBERTa [28], DistilBERT [51], XLNet [62] as well as S-MPNet [52], S-GTR-T5 [48], S-DistilRoBERTa and S-MiniLM [59]. 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.
- We assess their blocking effectiveness, efficiency and scalability.
- We compare their relative performance for both supervised and unsupervised matching.
- We demonstrate that high performance can be achieved by an end-to-end solution that leverages the best language model both for blocking and matching in many datasets with either long or short textual attributes.

2 Related Work

Blocking. The first approach to leverage embeddings vectors for blocking is DeepER [13], which uses GloVe for the vectorization of entities and hyperplane LSH for indexing and querying. AutoBlock [65] goes beyond DeepER by leveraging FastText embeddings in combination with cross-polytope LSH. Yet, it treats blocking as a classification task, requiring a large number of labelled instances to train its bidirectional-LSTM. DeepBlocker [55] 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. Given that efficiency is crucial for blocking, DeepBlocker with Auto-Encoder and FastText 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 [13], which leverages GloVe. Its generalization, DeepMatcher [33], is a framework for deep learning-based matching algorithms that supports the main pre-trained static models, i.e., GloVe and FastText, with the last one being the predefined option. Similarly, GraphER [23], Seq2SeqMatcher [35], CorDEL [60], MCAN [64], HierMatcher [14] and HIF-KAT [63] 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, context-aware nature. The first such work is EMTransformer [3], which considered BERT and its three main variants: XLNet, RoBERTa and DistilBERT. The same models are used by GNEM [5], which extends EMTransformer through a graph that captures the relations between all candidate pairs that are given as

Model	Dim.	$\|Seq.\|$	Param.	Blocking	Matching
Word2Vec (WC)	300	-	-	[55]	[33]
FastText (FT)	300	-	-	[55, 65]	[23, 33, 35], [14, 60, 63, 64]
GloVe (GE)	300	-	-	[13, 55]	[13, 33]
BERT (BT)	768	100	110M	-	[3, 5, 25, 38, 45]
AlBERT (AT)	768	100	12M	-	[38]
RoBERTa (RA)	768	100	125M	-	[3, 5, 25, 38]
DistilBERT (DT)	768	100	66M	-	[3, 5, 25, 38]
XLNet (XT)	768	100	110M	-	[3, 5, 25, 38]
S-MPNet (ST)	768	384	110M	-	-
S-GTR-T5 (S5)	768	512	110M	-	-
S-DistilRoBERTa (SA)	768	512	-	-	-
S-MiniLM (SM)	384	256	22M	-	-

Table 1: The language models used in our experiments.

input to matching. GNEM also applies this idea to DeepMatcher, in combination with FastText embeddings. DITTO [25] extends EM-Transformer by combining the BERT-based language models with external, domain-specific information (e.g., POS tagging) and data augmentation, which provides more (synthetic) training instances.

JointBERT [45] 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 [38], 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 [37], but it disregards all other models as well as the task of blocking. Surveys [27], books [47] and tutorials [56] about language models are too generic, without any emphasis on ER, and thus, are orthogonal to our work. Our goal in this work is to cover these important gaps in the literature.

Sentence-similarity tasks in NLP. Semantic Textual Similarity (STS) is an important NLP task, that is mostly evaluated in Transformer Models with the STS-B task [4] via the GLUE Benchmark [58]. Since in schema-agnostic ER all attributes in a record are concatenated into a "sentence", these tasks seem related. However, the characteristics of the input data differ. For instance, popular STS benchmarks contain sentences like image captions or news headlines, whereas typical ER benchmarks like the datasets considered here contain attributes such as person or product names, movie titles, addresses, etc. Concatenating such attributes does not form actual sentences, even if they are treated as such. Moreover, the task in STS is to predict a similarity score that indicates how similar the meaning of two sentences is, whereas in ER it is to decide whether two entity instances refer to the same real-world entity or not. Hence, it is not safe or straightforward to assume that language models will exhibit the same performance in ER as in STS.

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 (https://www.sbert.net/docs/pretrained_models.html), 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 the ER works that have used it for blocking or matching. Below, we briefly describe the selected models per category in chronological order. Most models have several versions (e.g., base, large) that differ in the number of learned parameters. To ensure a fair comparison, we consider the base version of each model. We also conducted some tests with larger versions, without observing notable differences in the results.

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 with low-dimensional dense ones of fixed size, which define a mathematical space, where semantically similar words tend to have low distance.

Word2Vec [31, 32] 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 [46] 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 [57], which are a natural evolution of Encoders-Decoders [54]. 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 [11], 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. NSP is sentence-based, since it receives a first sentence as input and tries to predict whether a second sentence can follow it.

AlBERT [22], which stands for "A little BERT", is a lighter version of BERT. BERT-base 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 [28] 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 [28].

DistilBERT [51], which stands for "Distillation BERT", is a lighter version of BERT that uses distillation [16, 50]. 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 [62] 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 [49]. SBERT [49] 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 [52], 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-GTR-T5 extends GTR [34], which is a dual encoder that encodes two pieces of text into two dense vectors respectively. This is typically used to encode a query and a document to compute their similarity for dense retrieval. GTR models are built on top of T5 [48], 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 [59], which distills BERT to produce a much lighter student. Unlike DistilBERT and other distillation strategies [17, 53], 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 the computationally expensive fine-tuning of the teacher.

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[1], contains descriptions of restaurants. D_2 contains products extracted from two online retailers, Abt.com and Buy.com [20]. D_3 comes from the same domain, matching products from Amazon.com and the Google Base data API (Google Pr.)[20]. D_4 involves bibliographic data from two publication repositories, DBLP and the ACM digital library [20]. 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) [36]. D_8 is another dataset from the product matching domain, involving descriptions from Walmart and Amazon [33]. Similar to D_4, D_9 contains bibliographic data from DBLP and Google Scholar [20]. Finally, D_{10} matches movies from IMDb and DBpedia [41], but has no overlap with the IMDb data source of D_5 and D_6.

All these datasets are publicly available through Zenodo[2] in CSV format. Their detailed characteristics are shown in Table 2(a). Note

[1] http://oaei.ontologymatching.org/2010/im
[2] https://zenodo.org/record/6950980

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{s_1}	D_{s_2}	D_{s_3}	D_{s_4}	D_{s_5}	D_{s_6}	D_{s_7}
Dat_1 Dat_2	$Rest_1$ $Rest_2$	Abt Buy	Amz GPr.	DBLP ACM	IMDb TMDb	IMDb TVDB	TMDb TVDB	Wmt Amz	DBLP Scholar	IMDb DBP	D_{10K}	D_{50K}	D_{100K}	D_{200K}	D_{300K}	D_1	D_{2M}
$\|V_1\|$ $\|V_2\|$	339 2,256	1,076 1,076	1,354 3,039	2,616 2,294	5,118 6,056	5,118 7,810	6,056 7,810	2,554 22,074	2,516 61,353	27,615 23,182	10K	50K	100K	200K	300K	1M	2M
$\|A_1\|$ $\|A_2\|$	7 7	3 3	4 4	4 4	13 30	13 9	30 9	6 6	4 4	4 7	12	12	12	12	12	12	12
$\|D\|$	89	1,076	1,104	2,224	1,968	1,072	1,095	853	2,308	22,863	8,705	43,071	85,497	172,403	257,034	857,538	1,716,102
$\|\bar{S}\|$	18.67	198.64	792.43	133.29	81,49	71.48	104.16	103.35	115.57	54.04	84.32	84.21	84.34	84.30	84.30	84.31	84.32

(a) Clean-Clean ER **(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.

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 [10, 42].

Datasets for Blocking Scalability. To evaluate blocking scalability, we employ the datasets shown in Table 2(b), which are widely used in the literature for this purpose [8, 19, 44]. These correspond to *Dirty ER*, a.k.a., Deduplication, where a single data source containing duplicates is given as input [10, 42]. They were artificially generated by Febrl [6], 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, resp.

Datasets for Supervised Matching. For this task, we used the same five datasets as in [3], which are widely used in the literature [25, 33]. 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. 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 so as to increase their difficulty [25].

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. These *schema-agnostic* settings inherently address misplaced attribute values (e.g., cases where person names are associated with their profession), while exhibiting high effectiveness both in blocking [39, 44] and matching [3].

All our code and datasets used in this experimental analysis are also publicly available in the above repository. For the language models, we used the implementations provided by two highly popular Python packages: Gensim[3] and Hugging Face[4]. The former offers Word2Vec and FastText, while the latter provides all other models. All experiments were executed on a server with Ubuntu 20.04, AMD Ryzen Threadripper 3960X 24-Core processor, 256 GB RAM and an RTX 2080Ti GPU. The GPU is used where possible, i.e., for vectorization of the dynamic models, similarity score calculation in matching and blocking, nearest neighbor search in blocking, and fine-tuning of the dynamic models in supervised matching.

4.3 Evaluated Tasks and Methodology

We evaluate the language models in the following four tasks.

Vectorization. This converts every given textual entity into its embedding vector. We consider schema-agnostic ER, where each entity is represented by a "sentence" that 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. In this task, we compare the execution time for each model.

Blocking. Given the vectorized entities of the input datasets, blocking produces a set of candidate pairs. For each input entity, we perform a *nearest neighbor search* (NNS) to find the k most similar vectors to it. For the datasets for Clean-Clean ER in Table 2(a), we perform exact NNS. For each entity in the smallest of the two datasets, we compute all similarity scores and return the k nearest neighbors. For the datasets for Dirty ER in Table 2(b), which are significantly larger, we perform approximate NNS. According to the state of the art, we follow [24], leveraging an HNSW [29] index. This is a graph-based index, serving as an approximation to a Delaunay graph, but with long-range links as well, to support the small-world navigation property. To avoid the connectivity issues raised by high-degree nodes in the original NSW, HNSW introduces a multi-layer separation of links based on their degree as well as on an advanced heuristic for better selecting the neighbors per node. The resulting index offers very good querying times with the trade-off of a large overhead in building the index. In our experiments, we used the implementation provided by FAISS.[5] First, we vectorize and index all input entities. Then, we query the index with every

[3] https://radimrehurek.com/gensim
[4] https://huggingface.co
[5] https://faiss.ai/cpp_api/struct/structfaiss_1_1IndexHNSW.html

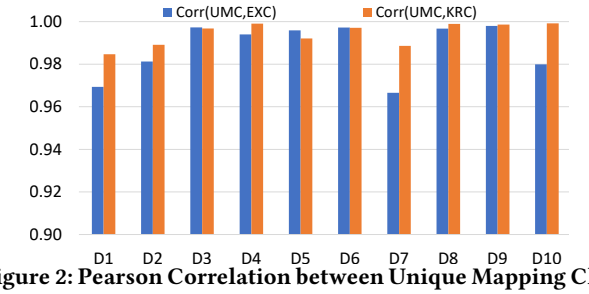


Figure 2: Pearson Correlation between Unique Mapping Clustering (UMC), Exact Clustering (EXC) and Kiraly Clustering (KRC).

entity e to retrieve its k approximate nearest neighbors in terms of Euclidean distance.

Unsupervised Matching. This is considered a clustering task, where each cluster of entities corresponds to a different real-world object. For two datasets in Clean-Clean ER, we model the task as bipartite graph matching, where each entity from the one dataset is matched with at most one entity from the other. To solve this, we apply *Unique Mapping Clustering* (UMC) [21], which achieves both high effectiveness and time efficiency [40].

The entire process is as follows. First, we calculate the similarity score between all pairs of entities using the following formula: $sim(e_i, e_j) = 1/(1 + dist(v_i, v_j))$, where $dist$ denotes the Euclidean distance between the embedding vectors v_i and v_j of the entities e_i and e_j, respectively. We do not perform blocking here to avoid its impact on the effectiveness of UMC. As execution time here we measure exclusively the run-time of UMC, i.e., assuming that all similarity scores have been computed already. UMC iterates over all pairs in descending order of similarity score, until all entities from the smallest dataset have been matched, or there are no more pairs that exceed a given similarity threshold δ. This threshold is the only configuration parameter of UMC. To fine-tune it, we consider all values in $[0.05, 0.95]$ with a step of 0.05 and select as optimal the one maximizing F-measure. In this way, our experimental analysis considers the maximum effectiveness per language model, comparing their potential. In practical settings, though, specifying the optimal similarity threshold for UMC is a non-trivial task.

To ensure generality of our results, besides UMC we also tested two other highly-performing algorithms from [40]: *Exact Clustering*, which matches two entities if they are mutually the best matches, and *Kiraly Clustering*, which provides a linear time approximation to the maximum stable marriage problem. In both cases, the results exhibited very high (>0.9) Pearson correlation with those of UMC. This can be seen in Figure 2.

Supervised Matching This is considered a binary classification task, classifying each candidate pair as `match` or `non-match`. Typically, a training, validation and testing set are used to learn the classification model, choose its optimal configuration, and assess its performance on new instances, respectively.

To examine the performance of BERT and SentenceBERT models, 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 [25], which leverages external information, such as POS tagging. However, EMTransformer is incompatible with the static models. To address this issue, we combine them with the state-of-the-art approach for this type of models, namely DeepMatcher [33].

Note that this analysis includes models that are supported by EMTransformer or DeepMatcher with minor adjustments to the code. S-GTR-T5 and Word2Vec are thus excluded, since the existing implementations could not support them (EMTransformer cannot handle the sequence2sequence input required by the former, while the latter is not in the format required by DeepMatcher). Note also that 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 [26]. 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 [25, 33]. For this analysis, we used the five datasets shown in Table 3.

5 Comparison on Effectiveness

5.1 Blocking

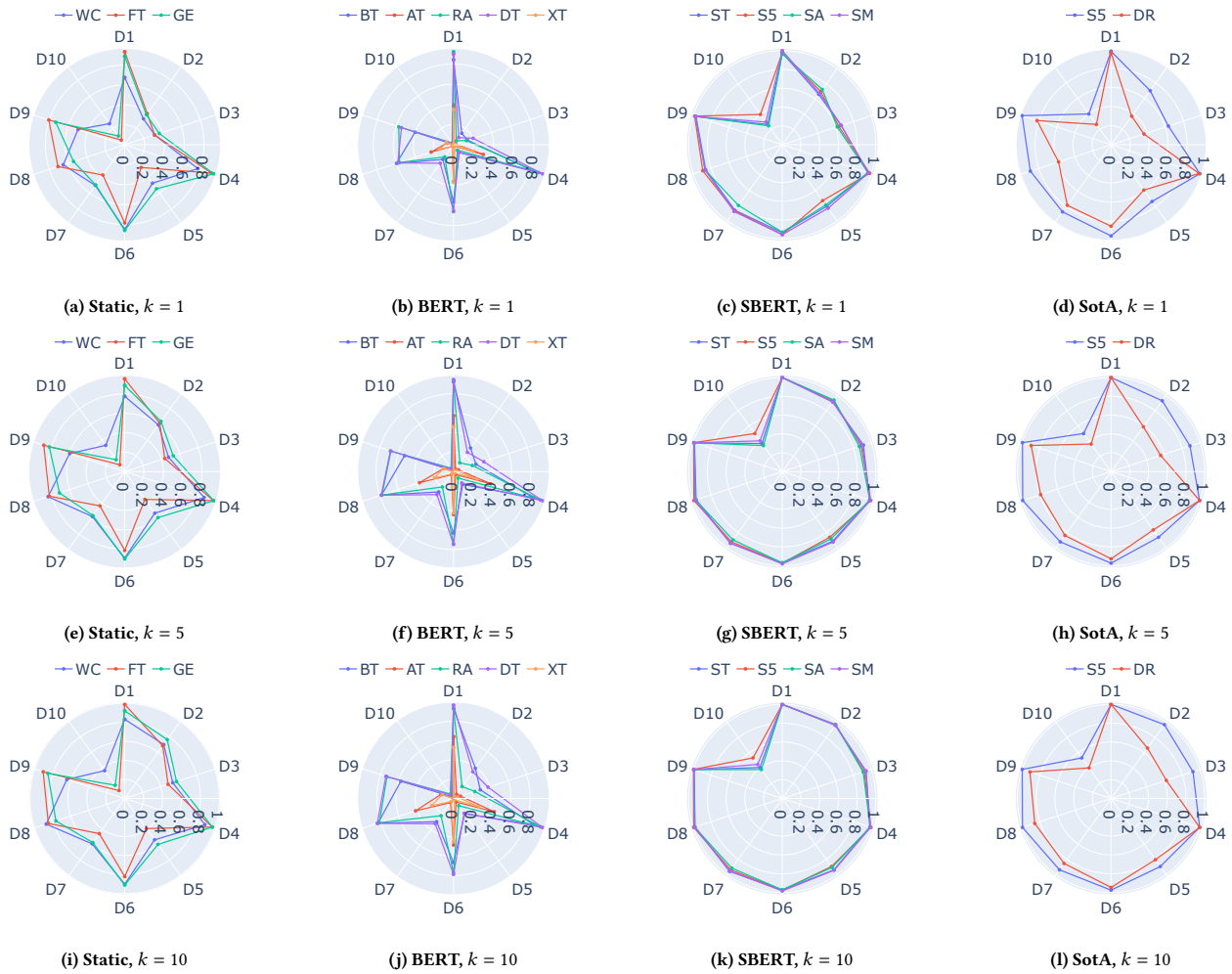
We measure the recall of the resulting candidate pairs, which is also known as pairs completeness [8, 19, 39]. Recall is the most critical evaluation measure for blocking, as it typically sets the upper bound for the subsequent matching step, i.e., a low blocking recall usually yields even lower matching recall, unless complex and time-consuming iterative algorithms are employed [7, 15]. In contrast, precision is typically low after blocking, due to the large number of false positives, but significantly raises after matching. Thus, in terms of precision, all models have the same denominator (i.e. total number of candidates) and precision can be omitted, since recall and precision behave the same.

The experimental outcomes are shown in Figure 3 for each category of models, for $k = 10$. More specifically:

In static models, GloVe is the top performer in the vast majority of cases, leaving FastText and Word2Vec in the second and third place, respectively. On average, GloVe outperforms FastText by 19%, except for D_1, D_8 and D_9, where FastText takes the lead. Compared to Word2Vec, GloVe's recall is higher by 12.5%, on average, except for D_7 D_8 and D_{10} (which abounds in noisy and missing values).

Among the BERT models, we can see that XLNet and AlBERT have very poor performance on almost all datasets. XLNet relies on permuted language modeling (PLM), which tries to model dependencies between words and phrases, instead of masked language modeling (MLM), which is adopted in BERT. This proves to be less effective in our task, where the input text is constructed by concatenating several different attributes, thus not constituting a coherent sentence. AlBERT trains only one encoder to produce a lighter version of BERT and shares its weights with the remaining encoders. This also turns out to perform poorly here. As a result, both models suffer from poor discriminativeness, i.e., they assign low similarity scores to both matching and non-matching pairs of entities. For the same reason, albeit to a lesser extent, the same applies to the remaining BERT models, which have a mediocre performance, with DistilBERT being the best one in all datasets.

Finally, all SentenceBERT models achieve very high recall across all datasets, but the extremely noisy and sparse D_{10}. The best model in this category is S-GTR-T5. This has to do with the base model that each SentenceBERT model relies to. For example, GTR-T5 trains in



(a) Static, $k = 1$ (b) BERT, $k = 1$ (c) SBERT, $k = 1$ (d) SotA, $k = 1$

(e) Static, $k = 5$ (f) BERT, $k = 5$ (g) SBERT, $k = 5$ (h) SotA, $k = 5$

(i) Static, $k = 10$ (j) BERT, $k = 10$ (k) SBERT, $k = 10$ (l) SotA, $k = 10$

Figure 3: Blocking recall per model across all datasets in Table 2(a). Each line of plots corresponds to a value of $k \in \{1, 5, 10\}$.

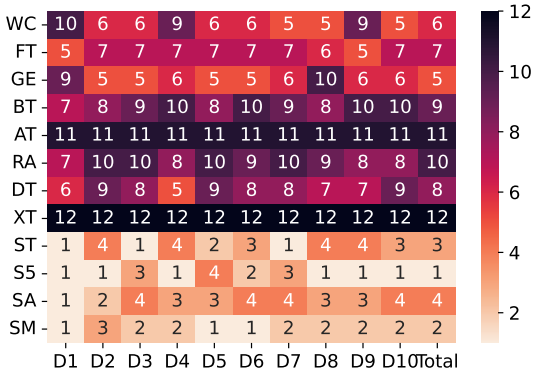


Figure 4: Model ranking wrt blocking recall (lower is better).

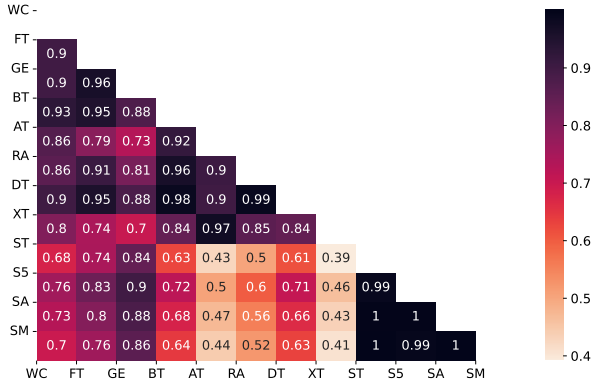


Figure 5: Pearson correlation of models wrt blocking recall.

a dataset with more than 2B pairs, while all three other base models train on a collection of datasets that amount to 1B+ pairs.

Finally, between groups, we can see that the point made in [49] holds in our task as well. BERT models, if not fine-tuned, behave worse in terms of recall than static models (GloVe) and Sentence-BERT have overall the best performance. There are two main reasons for the latter: they are designed for sentence rather than word embeddings and they are trained on wider corpora.

Summary. The above patterns are summarized in Figure 4, which reports the ranking of each model per dataset with respect to recall

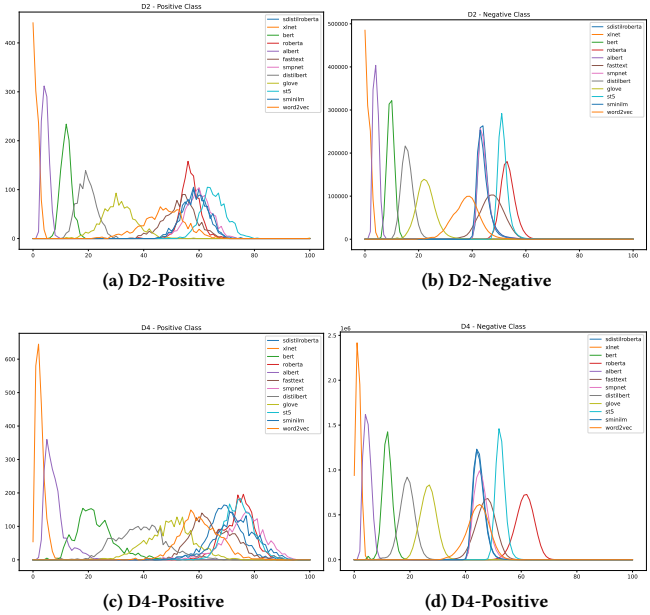
for k=10, with the rightmost column indicating the average position per model. We observe that the first four places are occupied by the SentenceBERT models, with S-GTR-T5 ranking first in most datasets. It is interesting that none of these models falls below the fourth place. There are two reasons for the superiority of SentenceBERT models: (a) They are inherently capable of transforming a sentence into an embedding vector, unlike the other two types, which are crafted for vectorizing individual tokens. (b) They encapsulate knowledge from wider corpora, while their final layer comes with reasonable weights already in its pre-trained form (unlike the BERT models).

The next three ranking positions mostly correspond to the static models, with Glove having the highest average one. Yet, FastText is the most stable one, fluctuating between positions 5 and 7, unlike the 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. The BERT models underperform the static ones, because they suffer from poor discriminativeness, due to the lack of fine-tuning, which guarantees their context-aware functionality. The predetermined weights in their final layer yield very low scores to most pairs of entities, regardless of whether they are matching or not. This is not true for the static models, despite their context-agnostic functionality and their lower dimensionality.

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 5. 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.

To a greater or lesser extent, all BERT-based models suffer from poor discriminative power in the tasks of blocking and unsupervised matching. This can be attributed to the lack of fine-tuning, which guarantees their context-aware functionality. The predetermined weights in their final layer yield very low scores to most pairs of entities, regardless of whether they are matching or not. This is not true for the static models, despite their context-agnostic functionality and their lower dimensionality. This is shown in Figure 6, which depicts the distribution of similarity scores among datasets D2 and D4 per language model (the positive class per dataset appears in the left column and the negative one in the right). We observe high discriminativeness for SentenceBERT models, lower for the static ones, while the BERT ones fails to separate the two distributions.

Comparison to SotA. The rightmost column in Figure 3 compares the best performing language model, S-GTR-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. S-GTR-T5 consistently outperforms DeepBlocker's recall to a significant extent. The only exceptions are D_1 and D_4,



(a) D2-Positive (b) D2-Negative

(c) D4-Positive (d) D4-Positive

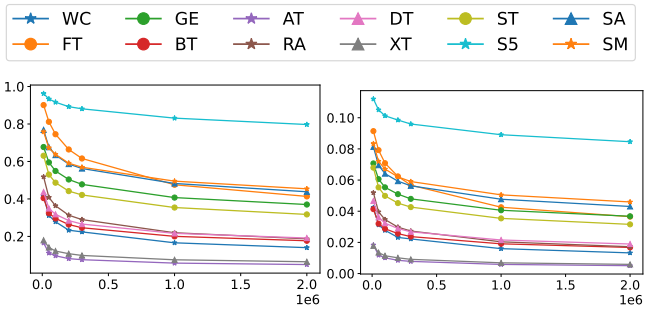
Figure 6: Discriminativeness for all models for cases D2 and D4 in the classes of Match (Positive) and Non-match (Negative).

where both methods achieve practically perfect recall 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, S-GTR-T5's recall is 15% higher than DeepBlocker.

Seemingly, this can be attributed to the FastText embeddings used by DeepBlocker. However, DeepBlocker is a comprehensive blocking method that uses FastText in a more complex way than the mere nearest neighbour search of S-GTR-T5. For example, a crucial component of DeepBlocker is self-supervision, which automatically labels a random sample of the candidate pairs in order to train its classification model. As a result, DeepBlocker is a stochastic approach, unlike S-GTR-T5, which exclusively performs nearest neighbor search. The ablation analysis in [55] indicates that all DeepBlocker's components have a significant contribution to the final outcome. For this reason, the role of the language models is restricted and, thus, the performance of DeepBlocker exhibits a low correlation with that of FastText+NNS.

5.1.1 Scalability. 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 in Table 2(b). The results appear in Figure 7.

Figure 7(a) shows 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-GTR-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%. S-GTR-T5 has been trained on a much larger and



(a) Recall (b) Precision

Figure 7: Scalability effectiveness over the synthetic datasets in Table 2(b). The horizontal axis indicates the number of input entities.

richer corpus. The advantage that this offers to S-GTR-T5 becomes much more evident when testing it on the synthetic datasets, which are more challenging due to the much larger number of candidate pairs. 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 lie 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 7(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).

5.2 Unsupervised Matching

Figure 8 reports the performance of all models in this task.

In static models, based on the average distance from the maximum f-measure (F1), 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.

In the BERT-based 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. The reason is the same as in Blocking: both were trained for a different task and cannot perform well in the task of Matching, without further fine-tuning.. 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.

In the SentenceBERT models, the best one is S-GTR-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%.

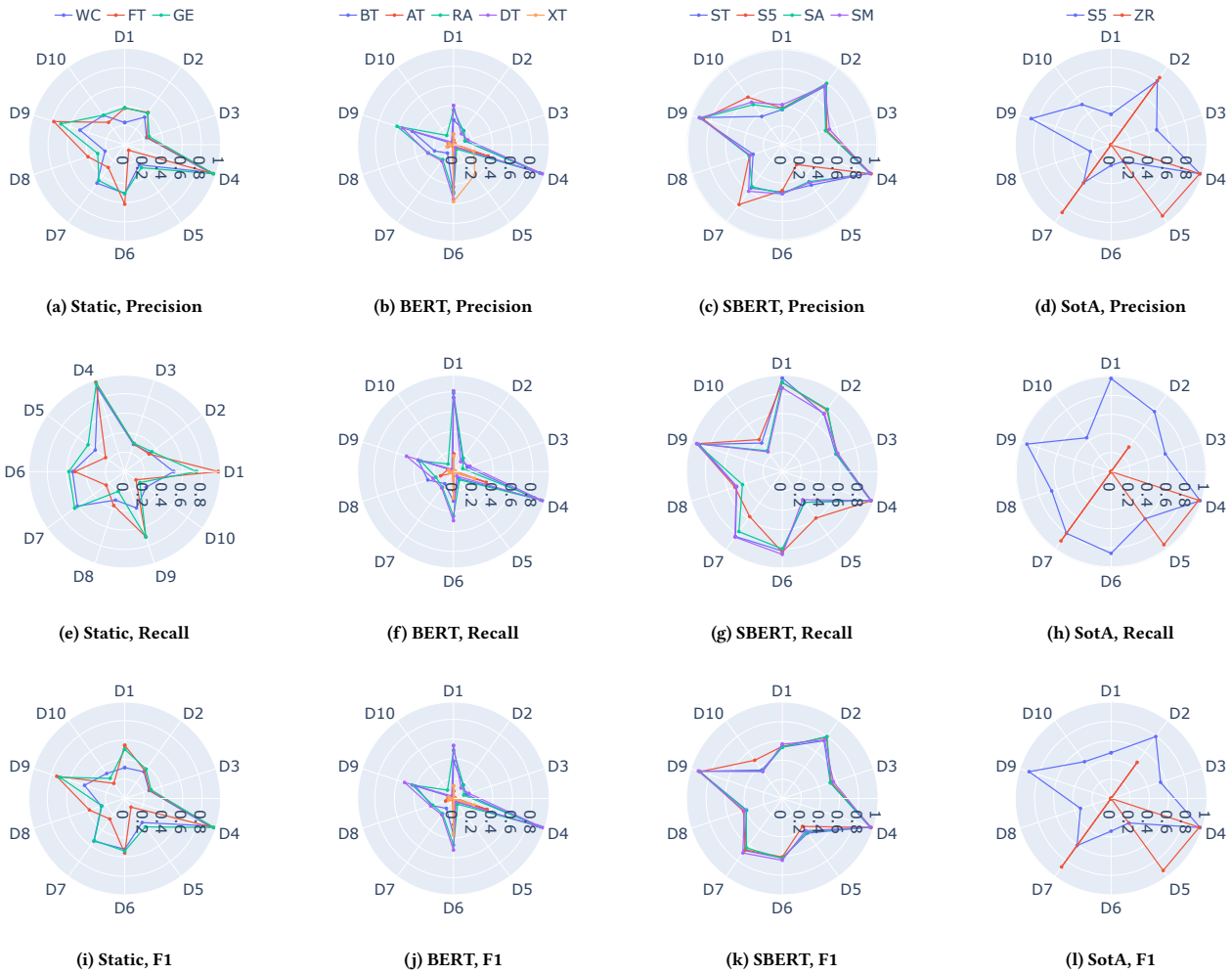
In absolute terms, their best performance corresponds to the bibliographic datasets, D_4 and D_9, where their F1 remains well over 0.9. The reason is that the long titles and list of authors facilitate the distinction between matching and non-matching entities. Also high (~0.8) is their F1 over D_2, 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.

Finally, it is worth stressing that as in Blocking, S-GTR-T5 outperforms the best models of the other two types, since all other models do not surpass 0.5 in F1 – except for the relatively clean and easy D_4. In contrast with Blocking, though, in Matching we can perform fine-tuning to check whether BERT models can perform better than other two types. See Section 5.3 for more details.

Summary. Figure 9 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 10 shows an almost perfect dependency between the SentenceBERT 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.

Overall, the relative performance of the three model types follows the same patterns as in Blocking, due to the same root causes. The SentenceBERT models outperform all others, as they are inherently crafted for vectorizing entire "sentences", while they have been trained on much larger corpora. At the other extreme lie the BERT models, which lack fine-tuning, with the predefined weights in their final layer yielding low similarities for matching and non-matching pairs alike. The static models lie in the middle of these extremes, due to their context-agnostic, word-level embeddings.

Comparison to SotA. The state-of-the art in Unsupevised Matching is ZeroER [61], which converts every pair of entities into a feature vector whose dimensions correspond to similarity functions. At its core lies the assumption that the resulting feature vectors are generated by a Gaussian Mixture Model with two mixture components (one for each matching category). Adaptive feature regularization is leveraged to avoid overfitting, while transitivity



(a) Static, Precision (b) BERT, Precision (c) SBERT, Precision (d) SotA, Precision

(e) Static, Recall (f) BERT, Recall (g) SBERT, Recall (h) SotA, Recall

(i) Static, F1 (j) BERT, F1 (k) SBERT, F1 (l) SotA, F1

Figure 8: Precision, Recall and F1 per model across all datasets in Table 2(a).

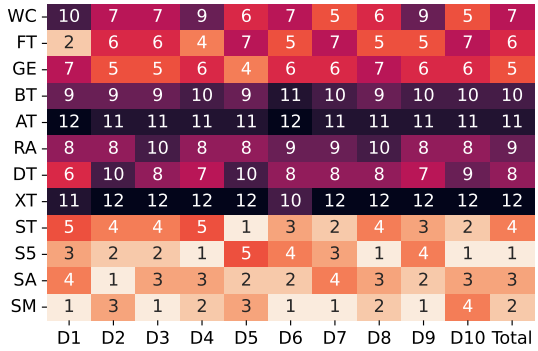


Figure 9: The ranking position of each model with respect to Unsupervised Matching F1 per dataset. Lower is better.

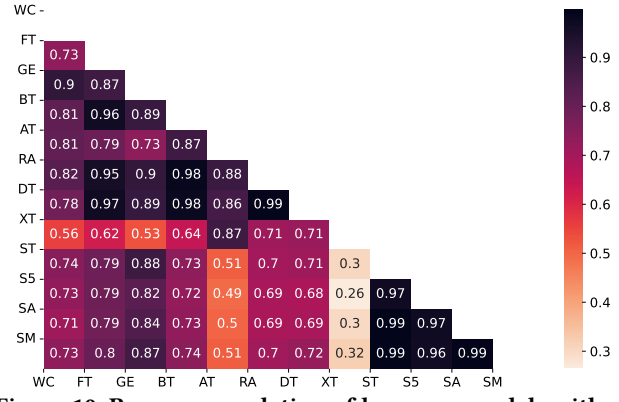
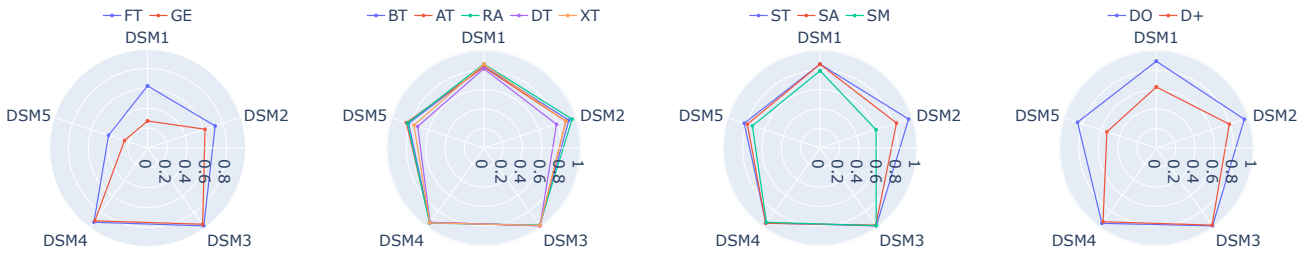


Figure 10: Pearson correlation of language models with respect to Unsupervised Matching F1.

improves its accuracy. ZeroER uses Magellan's overlap blocking to reduce the search space to a small set of candidate pairs.

We compare ZeroER with an end-to-end framework based on the best language model for Blocking and Matching, i.e. S-GTR-T5. We actually use the above matching algorithm with the similarity threshold set to 0.5 by default, but instead of utilizing all pairs of entities, every entity of the smallest entity collection is allowed only k=10 candidates, produced by Blocking with exact NNS.

The relative performance of the two approaches appears in Figure 8(d). ZeroER lacks an estimated performance for half the



(a) static **(b) BERT** **(c) SBERT** **(d) SotA**

Figure 11: Supervised Matching F-Measure per model across all datasets in Table 2(b).

datasets, because it did not terminate after 6 hours – unlike S-GTR-T5, which consistently takes less than 1 minute, as shown in Table 5(b). In D_4, both methods have the same, almost perfect performance, due to the rather clean data and the relatively easy task. In D_1 and D_2, S-GTR-T5 outperforms ZeroER to a significant extent. ZeroER actually yields F_1=0 on D_1, because D_1 contains many missing and misplaced values, which cannot be supported by ZeroER's schema-based functionality (unlike the schema-agnostic settings of S-GTR-T5). D_2 conveys large textual values, which are also barely suitable for most similarity measures employed by ZeroER. In contrast, D_5 and D_7 contain short attribute values that describe movies (e.g., actor names). These are ideal for the features of ZeroER, which thus achieves much higher effectiveness than S-GTR-T5, which is not crafted for rare, domain-specific (terminological) textual values.

Overall, S-GTR-T5 performs significantly better than or at least equally well as ZeroER in most datasets, despite its parameter-free functionality, while being orders of magnitude faster (cf. Section 6.3).

5.3 Supervised Matching

Figures 11(a)-(c) show the F1 of all models in this task over the datasets in Table 3. We observe that the language models can be distinguished into two groups according to their context awareness. The dynamic models consistently exhibit the highest performance. RoBERTa is actually the most robust model in terms of effectiveness. It achieves the highest F1 in most datasets, ranking first on average. In fact, its 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 maximum F_1 per dataset: BERT, AlBERT, XLNet, S-DistilRoBERTa, S-MiniLM.

Even the least effective dynamic 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 [25, 33]). 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 dynamic models ranges from 4.9% (DSM_1) to 17% (DSM_2).

The second group of models includes the static, context-agnostic ones, which perform relatively poorly, even though they are combined with the best configuration of DeepMatcher, i.e., the state-of-the-art algorithm for this task and type of models. GloVe and FastText are actually combined with a two layer fully connected ReLU HighwayNet classifier followed by a softmax layer in the classification module in combination with a hybrid model for the attribute similarity vector module. On average, GloVe and FastText underperform the top model per dataset by 22% and 37%, respectively. Only in DSM_3 and DSM_4, where all dynamic models exceed 0.95, the two static models exhibit high performance, with their F1 within 5% of the maximum one (this is another indication about the straightforward matching task posed by the bibliographic data of these two datasets). Note that FastText consistently outperforms GloVe across all datasets, since its character-level functionality is capable of addressing the out-of-vocabulary tokens that arise, due to the domain-specific terminology of each dataset, unlike GloVe.

Overall, *the static models underperform the dynamic ones in most cases, as reported in the literature [25], while the BERT-based models match the SentenceBERT ones, unlike for the previous ER tasks, due to the fine-tuning of their last layer. SentenceBERT models also benefit from fine-tuning, but to a lesser extent, probably because the sentence representing each entity is constructed in an ad-hoc manner, lacking any cohesiveness.*

Comparison to SotA. Figure 11(d) depicts the performance of the state-of-the-art supervised matching algorithms that leverage static and dynamic models, DeepMatcher+ [18] and DITTO [25]. We actually consider their optimized F1 that is reported in [25].

Comparing DITTO to the dynamic models, we observe that its F1 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 ≤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, whose effect is more clear when comparing DITTO to the language model at its core, i.e., RoBERTa. On average, across all datasets, the latter underperforms DITTO by 1.3%.

	WC	FT	GE	BT	AT	RA	DT	XT	ST	S5	SA	SM
Init	32.4	159.7	5.87	4.72	3.99	5.28	4.3	4.73	9.19	9.84	9.33	8.36
D1	0.0	0.2	1.9	2.6	2.4	2.3	1.3	4.0	1.1	1.1	0.7	0.5
D2	0.1	1.6	0.2	3.1	2.4	2.3	2.2	3.3	3.4	3.4	1.8	0.9
D3	0.9	9.6	0.4	10.1	6.7	6.3	8.6	8.3	10.3	12.4	5.8	2.3
D4	0.2	2.5	0.3	5.9	5.2	5.3	4.2	7.8	5.1	5.4	2.8	1.4
D5	0.4	3.8	0.4	13.6	13.0	12.9	8.7	20.3	10.7	12.1	6.0	3.2
D6	0.6	5.5	0.5	15.4	14.3	13.8	10.4	21.3	14.9	17.2	8.2	3.9
D7	0.4	3.6	0.4	11.9	11.2	11.4	8.0	17.7	9.7	10.4	5.3	2.8
D8	1.0	10.0	0.8	28.8	25.0	24.2	19.5	38.9	28.5	27.3	14.9	6.7
D9	2.4	27.7	1.9	73.4	66.0	65.5	49.9	99.9	58.0	61.5	31.4	16.0
D10	1.1	10.6	1.2	51.1	49.1	47.9	31.6	78.9	28.9	30.2	16.5	10.3

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

Comparing the static models to DeepMatcher+, we observe that its performance is almost identical with FastText in most datasets, because it leverages the same language model. Only in DSM_2 and DSM_5, DeepMatcher+ performs substantially better, by 9% and 23%, respectively. This should be attributed to its advantage over the original DeepMatcher algorithm, which DeepMatcher+ combines with transfer and active learning. Note that DeepMatcher+ consistently outperforms GloVe by 28%, on average. Note also that DeepMatcher+ underperforms the dynamic models in practically all cases. This is particularly true in DSM_1 and DSM_5, where the worst dynamic model (DistilBERT) exceeds DeepMatcher+ by ~20%. This verifies the superiority of dynamic models over the static ones in supervised matching, due to their fine-tuning, which optimizes the weights of their last layer to the data at hand.

6 Comparison on Efficiency

6.1 Vectorization

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 (or character n-grams) 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 into dense embeddings vectors (after the initialization). 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.

Among 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-GTR-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-GTR-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-GTR-T5 over D_8-D_{10}.

6.2 Blocking

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. 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-GTR-T5 are consistently the fastest and slowest models, respectively. However, this overhead time is negligible in comparison to the vectorization time in Table 4.

Comparison to SotA. Table 5(a) reports the run-times corresponding to the rightmost column in Figure 3. We observe that DeepBlocker is consistently faster than S-GTR-T5 for k=1 and k=5 in all datasets but D_{10}. The reason for the slow operation of S-GTR-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-GTR-T5 leverages 768-dimensional embeddings vectors, compared to 300-dimensional FastText vectors of DeepBlocker. Yet, for k=10, DeepBlocker is faster than S-GTR-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-GTR-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 scale poorly as the size of the input data increases: in D_{10}, S-GTR-T5 is faster by 1.5 times for $k\in\{1, 5\}$ and 3.7 times for k=10.

6.2.1 Scalability Figure 13(a) shows that the blocking time of all models scales superlinearly, but subquadratically: as the number of input entities increases by 200 times from D_{10K} to D_{2M}, the run-times increase by up to 1,435 times. With the exception of

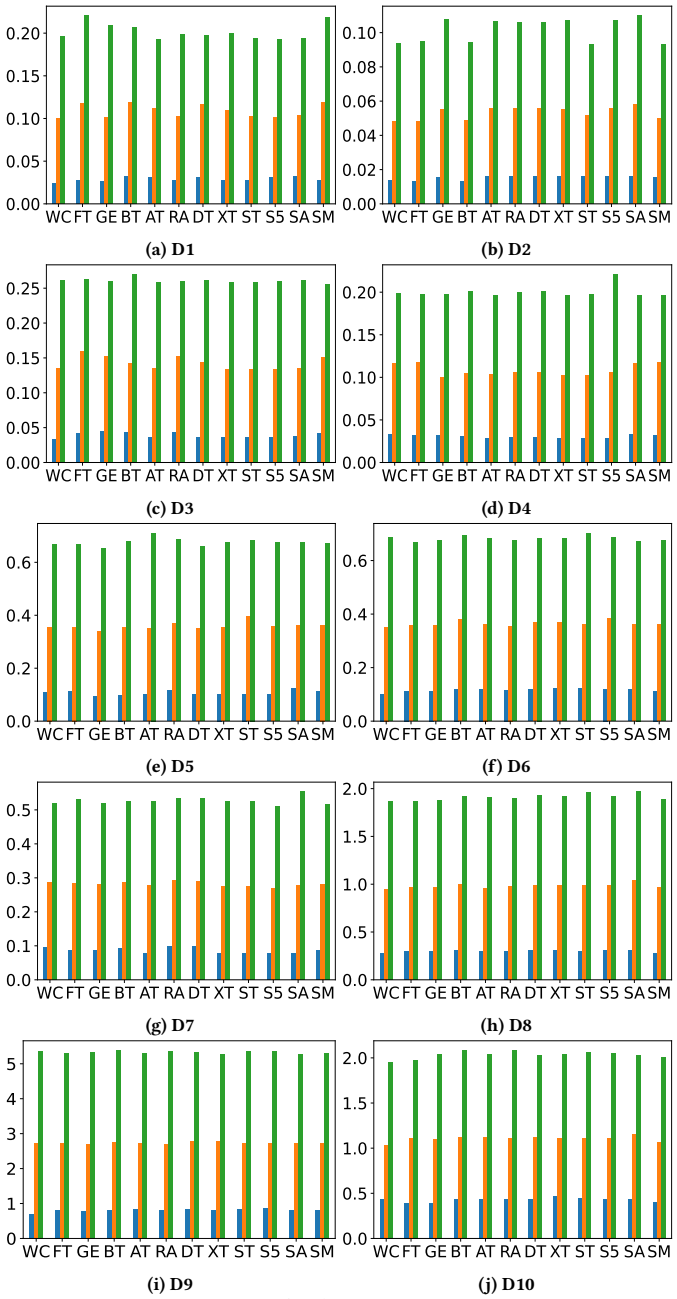


Figure 12: Blocking Time (sec) per method. There are three values for $k \in \{1, 5, 10\}$.

the two most efficient models, Word2Vec (742 times) and S-GTR-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, i.e., the former dominates the latter. 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 superlinearly.

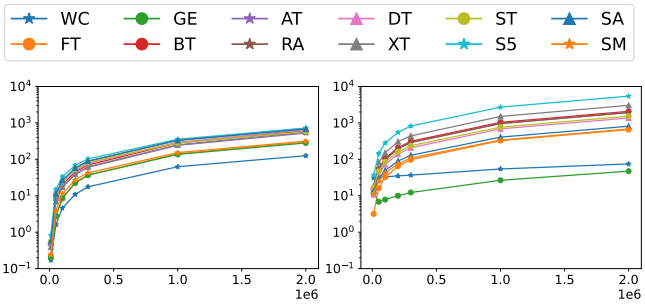


Figure 13: Scalability over the synthetic datasets in Table 2(b). The horizontal axis indicates the number of input entities.

	DeepBlocker			S-GTR-T5			ZeroER		S-GTR-T5	
	$k=1$	$k=5$	$k=10$	$k=1$	$k=5$	$k=10$	t_p	t_m	t_p	t_m
D1	8	8	17	11	11	11	2	1	13	1
D2	9	9	17	13	13	13	1,728	71	14	3
D3	19	19	36	22	22	23	-	-	23	4
D4	13	13	28	15	15	16	9,595	113	16	9
D5	24	25	63	22	22	23	1,291	10	23	12
D6	28	28	68	27	27	28	-	-	28	16
D7	22	22	49	20	22	21	1,599	338	21	13
D8	46	46	46	37	38	39	-	-	38	8
D9	111	111	270	72	74	77	-	-	72	10
D10	154	150	371	41	41	42	-	-	46	96

(a) Blocking **(b) Unsup. Matching**

Table 5: Comparison of S-GTR-T5 with (a) DeepBlocker in Blocking, and (b) ZeroER in Unsupervised Matching. All columns are in seconds, but the rightmost one which is in milliseconds. t_p (t_m) stands for preprocessing (matching) time.

Figure 13(b) reports the evolution of vectorization time, which increases sublinearly with the size of the input data for all models. 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-GTR-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). This should be attributed to the initialization of each model, which is independent of the input size and accounts for a large portion of the overall vectorization time of each model, as shown in Table 4. The larger the relative cost of initialization is, the lower is the increase in vectorization time as size of the input increase. As a result, 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 (by 205 times), due to its character-level functionality.

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-GTR-T5, which consistently exhibits the slowest

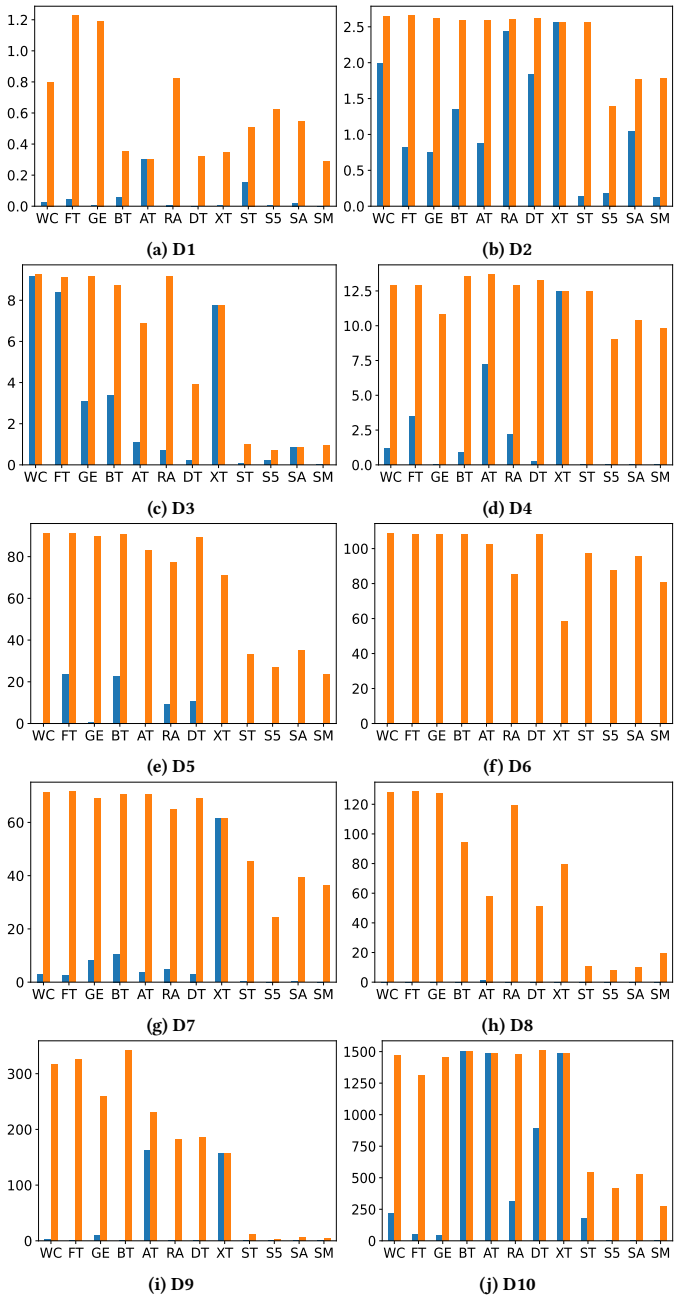


Figure 14: Unsupervised Matching Time (sec) per method. Blue is the time until the highest F1; orange is total time.

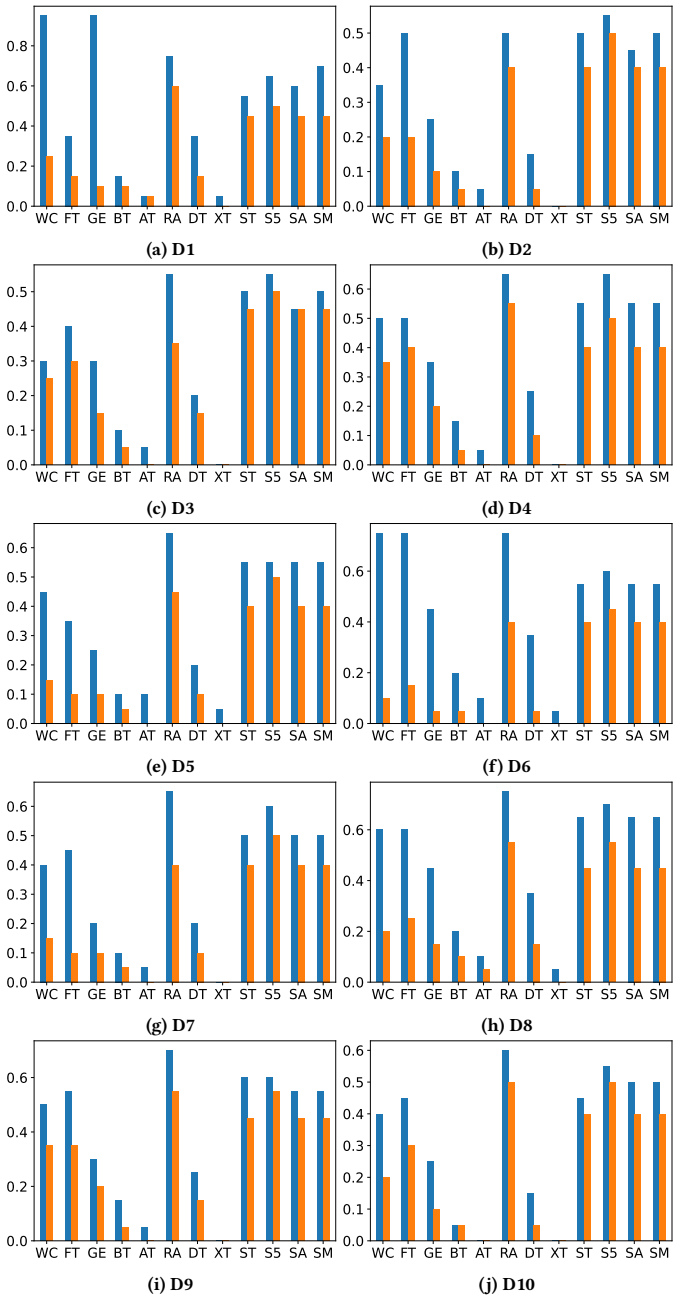


Figure 15: Unsupervised Matching δ per method. Blue is the δ where the best F1 is found and orange is the δ where the algorithm terminates.

vectorization, with XLNet being the second worst model across all datasets. For D_{2M}, they take 90.3 and 50.6 minutes, respectively.

6.3 Unsupervised Matching

We define as matching time of a model the time that is required for applying the UMC algorithm to all pairwise similarities using the optimal pruning threshold. For the most effective model, S-GTR-T5, the 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. Similar behavior is exhibited by the rest of the SentenceBERT models. The static models also apply UMC within few seconds in most datasets, depending on their similarity threshold. This is less frequently true for the BERT-based models, which suffer from low discriminativeness, thus yielding lower thresholds and significantly more pairs to be processed. As a result, one of the BERT-based models is the slowest one in most datasets – typically AlBERT or XLNet.

Comparison to SotA. Table 5(b) demonstrates that ZeroER's run-time is dominated by its blocking time, which is higher larger than the total execution time of S-GTR-T5 in most datasets. Due to its simple functionality, the end-to-end S-GTR-T5 approach is two orders of magnitude faster than ZeroER in all datasets, but D_1. Its run-time is dominated by the vectorization and the indexing of the input entities, with the matching time accounting for a few milliseconds even for the largest dataset, due to blocking.

6.4 Supervised Matching

Table 6 demonstrates 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 the top-performing model, RoBERTa, by 9% and 7%, respectively, on average. More significant gains in efficiency are achieved by the rest of the SentenceBERT models, S-DistilRoBERTa and S-MiniLM: they reduce RoBERTa's training and testing times by more than 50% in all cases, while their F1 is lower by just 5%, on average. Similarly, DistilBERT reduces RoBERTa's run-times to a half for a 7% reduction in F1. XLNet underperforms RoBERTa in all respects. XLNet is consistently the slowest by far model among all datasets, thus underperforming RoBERTa in all respects. The same applies to BERT, albeit to a minor extent, i.e., <2% with respect to all measures. ALBERT achieves slightly lower training times (by 7%) than RoBERTa at the cost of higher prediction times (by 8%), while its F1 is lower by just 2%, on average. Regarding the static models, on average, they are just 10% and 17% faster than RoBERTa with respect to training and testing time, respectively, despite their very low F1. Overall, we can conclude that *the SentenceBERT models are significantly faster than the BERT-base ones, thus achieving a better trade-off between effectiveness and time efficiency.*

	DSM1		DSM2		DSM3		DSM4		DSM5	
	t_t	t_e	t_t	t_e	t_t	t_e	t_t	t_e	t_t	t_e
FT	851.9	4.8	100.9	0.6	1,155.0	6.9	2,527.3	14.8	882.1	4.7
GE	847.0	4.8	101.4	0.6	1,157.2	6.9	2,534.3	14.7	876.9	4.7
BT	1,811.2	11.2	66.3	0.4	1,525.9	9.6	2,615.9	15.8	1,093.8	6.8
AT	1,700.5	12.0	62.8	0.5	1,448.4	10.4	2,422.1	17.0	1,026.6	7.4
RA	1,810.8	11.2	66.2	0.4	1,548.7	9.6	2,666.9	15.8	1,111.3	6.9
DT	915.2	5.5	34.0	0.2	779.2	4.8	1,341.9	7.9	559.1	3.4
XT	2,920.7	24.2	92.2	0.7	2,120.5	15.9	3,196.5	21.0	1,423.2	10.1
ST	1,667.5	10.5	63.4	0.4	1,475.8	8.8	2,549.9	14.4	1,076.8	6.3
SA	828.2	5.0	31.4	0.2	716.9	4.3	1,253.7	7.1	518.1	3.1
SM	406.6	2.1	13.4	0.1	299.2	1.7	521.6	2.7	216.6	1.2

Table 6: Training (t_t) and testing (t_e) times in seconds of all models in Supervised Matching over the datasets in Table 3.

7 Discussion & Conclusions

Figure 16 summarizes our experimental results on the three ER tasks. The horizontal axis in each diagram corresponds to the effectiveness measure of the respective task, while the vertical one corresponds to the normalized run-time, which is computed by dividing the overall run-time of a model with that of the fastest one (i.e., 1 is assigned to the fastest model). The space formed by these axes illustrates the trade-off between effectiveness and time efficiency, with the top performing model lying closer to (1,1), i.e., the lower right corner. Note that for each model, we have computed its average effectiveness and normalized time across all datasets in Table 2.

We observe that we can distinguish the ER tasks into two groups based on the relative performance of the three model types. The first group includes the unsupervised tasks, i.e., Blocking and Unsupervised Matching, where the SentenceBERT models consistently outperform the other two types to a significant extent. The reason is that they are able to distinguish between matching and non-matching entities without fine-tuning their top attention layers. Among them, the differences are minor, on average. Typically, though, S-GTR-T5 excels in effectiveness, but is slower, while S-MiniLM offers a better balance, trading slightly lower effectiveness for slightly higher time efficiency. The second best type includes the static models, where GloVe clearly dominates FastText and Word2Vec, which suffer from significantly higher initialization cost, while being slightly less effective. GloVe is actually the fastest model across all datasets in both ER tasks, thus being ideal for ER applications emphasizing time efficiency at the cost of lower effectiveness. Finally, all BERT-based models consistently underperform the other two types in all respects. Their poor effectiveness stems from the lack of fine-tuning, which causes them to assign low similarity scores to both matching and non-matching pairs alike.

Different patterns are observed in Supervised Matching, where all BERT-based models excel in effectiveness. As expected, DistilBERT sacrifices effectiveness for significantly higher run-time. Note that in this case, we exclusively consider the testing/prediction time per model, because the training time constitutes an one-off cost. S-MPNet also exhibits very high effectiveness, while S-DistilRoBERTa and S-MiniLM emphasize time efficiency. The former dominates DistilBERT, while the latter is the fastest albeit the least effective dynamic model. The static models are less accurate than all dynamic models, while offering no advantage in terms of run-time. Therefore, they should be avoided in this task. Instead, any of the dynamic models can be selected, depending on requirements of the application at hand. The only exception is XLNet, which is much slower but not more effective than most dynamic models.

In the future, we will extent our analysis on ER datasets with numeric values. We also intent to enhance our end-to-end, parameter- and learning-free approach to ER with SentenceBERT models, whose performance in Figure 8(d) is remarkable. We will explore ways of replacing its default thresholds with data-driven 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.

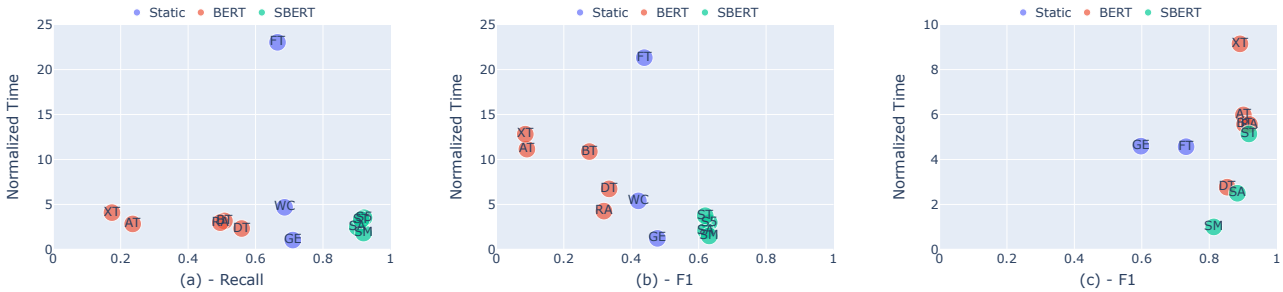


Figure 16: Tradeoff between Effectiveness and Time Efficiency on average, across all datasets in Table 2(a) for (a) Blocking with k=10 and (b) Unsupervised Matching (wrt best attainable F1) and all datasets in Table 2(b) for (c) Supervised Matching.

[4] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055* (2017).

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

[6] 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.

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

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

[9] 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.

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

[11] 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).

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

[13] 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.

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

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

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

[17] 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).

[18] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource Deep Entity Resolution with Transfer and Active Learning. In *ACL*. 5851–5861.

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

[20] 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.

[21] 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.

[22] 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).

[23] 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.

[24] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[25] 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.

[26] 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

[27] 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

[28] 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).

[29] 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.

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

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

[32] 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).

[33] 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.

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

[35] 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.

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

[37] 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.

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

[39] 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.

[40] George Papadakis, Vasilis Efthymiou, Emmanouil Thanos, and Oktie Hassanzadeh. 2022. Bipartite Graph Matching Algorithms for Clean-Clean Entity Resolution: An Empirical Evaluation. In *EDBT*. 2:462–2:474.

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

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

[43] 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.

[44] 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.

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

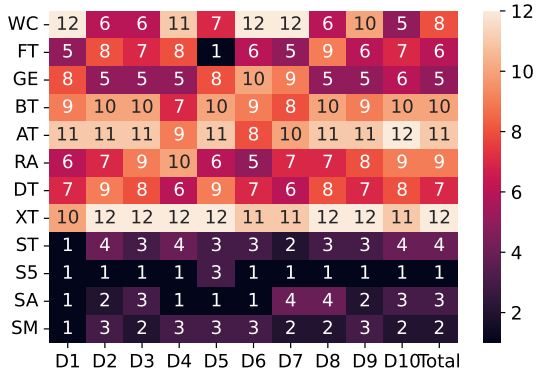


Figure 17: Method ranking wrt blocking recall (lower is better) (Schema-Based).

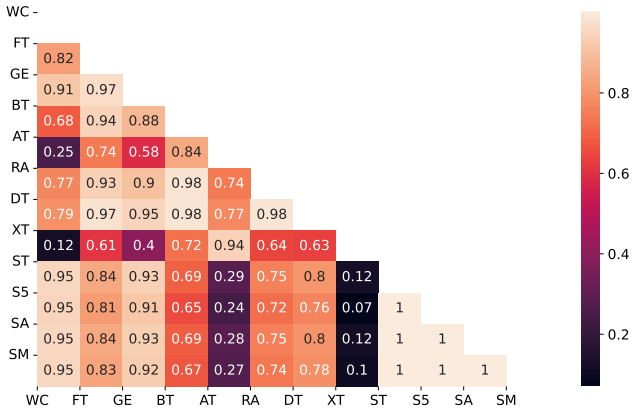


Figure 18: Pearson correlation of models wrt blocking recall (Schema-Based).

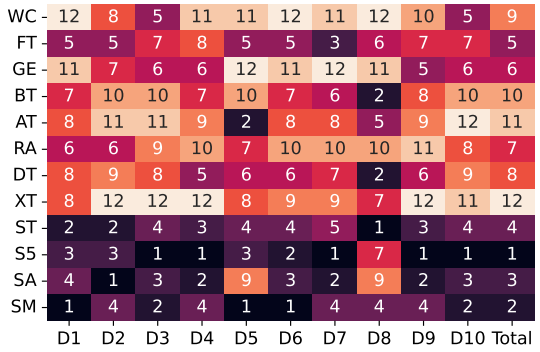


Figure 19: The ranking position of each method per dataset in unsupervised settings for Matching. Lower is better (Schema-Based).

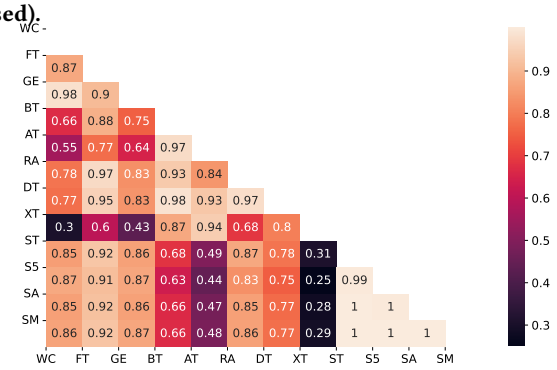


Figure 20: Pearson correlation with respect to F1 per pair of language models for unsupervised matching (Schema-Based).

[46] 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.

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

[48] 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.

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

[50] 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).

[51] 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).

[52] 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.

[53] 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).

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

[55] 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.

[56] 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.

[57] 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).

[58] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[59] 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.

[60] 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.

[61] 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.

[62] 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).

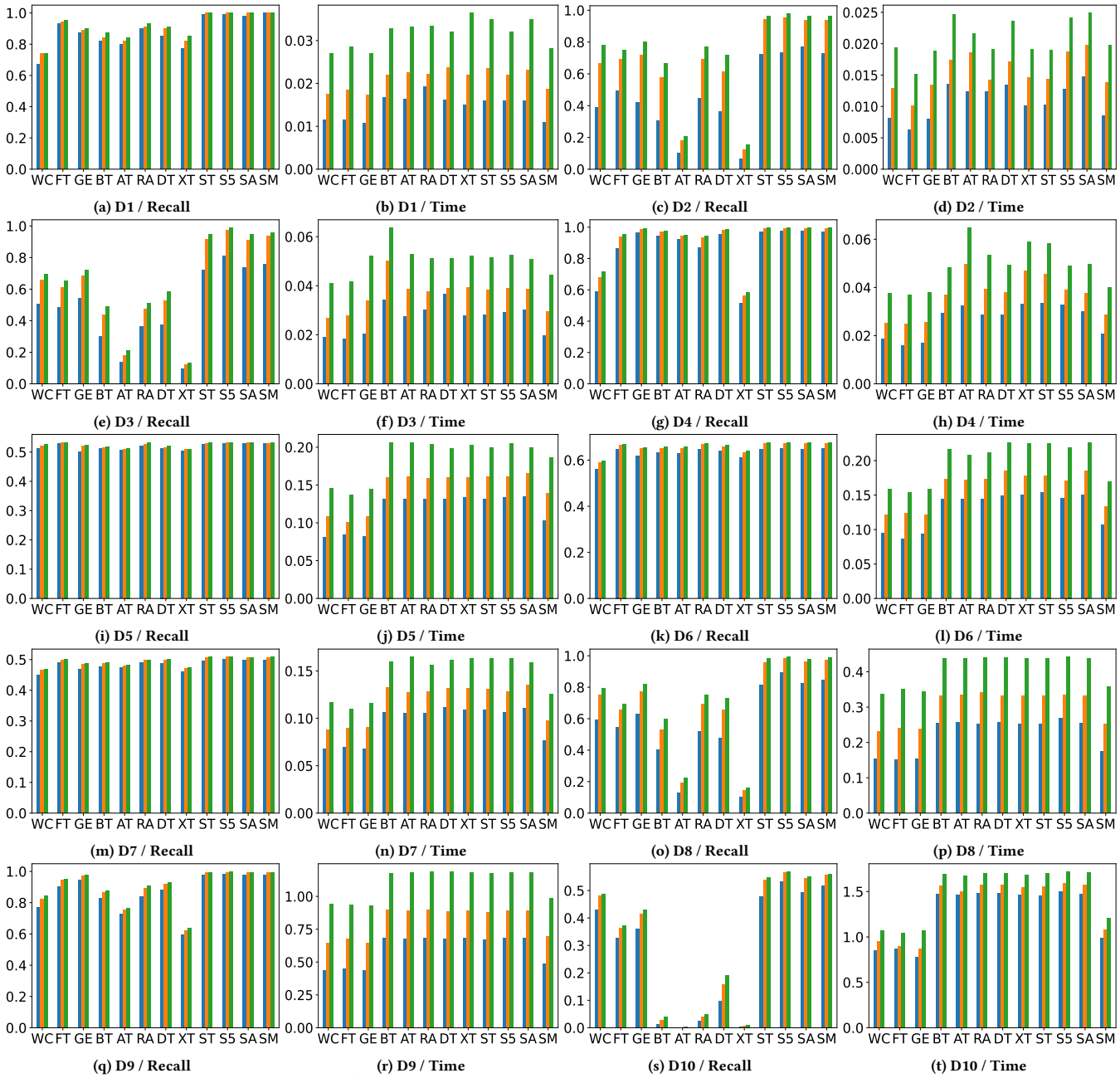
[63] Zijun Yao, Chengjiang Li, Tiansi Dong, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Yichi Zhang, and Zelin Dai. 2021. Interpretable and Low-Resource Entity Matching via Decoupling Feature Learning from Decision Making. In *ACL/IJCNLP*. 2770–2781.

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

[65] 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.

8 Appendix I: Schema-based Experiments

In Figures 17, 18 and 21 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) 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 21: 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 similar trend is followed in Figures 22, 20 and 19 for Unsupervised Matching.

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