EM-Join: Efficient Entity Matching Using Embedding-Based Similarity Join

Douglas Rolins Santana[1], Paulo Henrique Santos Lima[2] and Leonardo Andrade Ribeiro[2]
[1]*Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG), Goiânia, GO, Brazil*
[2]*Instituto de Informática (INF), Universidade Federal de Goiás (UFG), Goiânia, GO, Brazil*

Keywords: Data Cleaning and Integration, Deep Learning, Entity Matching, Experiments and Analysis.

Abstract: Entity matching in textual data remains a challenging task due to variations in data representation and the computational cost. In this paper, we propose an efficient pipeline for entity matching that combines text preprocessing, embedding-based data representation, and similarity joins with a heuristic-driven method for threshold selection. Our approach simplifies the matching process by concatenating attribute values and leveraging specialized language models for generating embeddings, followed by a fast similarity join evaluation. We compare our method against state-of-the-art techniques, namely Ditto, Ember, and DeepMatcher, across 13 publicly available datasets. Our solution achieves superior performance in 3 datasets while maintaining competitive accuracy in the others, and it significantly reduces execution time—up to 3x faster than Ditto. The results obtained demonstrate the potential for high-speed, scalable entity matching in practical applications.

1 INTRODUCTION

Entity matching (EM) is a critical step in data integration, aiming to identify records that refer to the same real-world entity within or across datasets. The task is challenging with textual data due to misspellings, format variations, and incomplete information. Traditional EM methods, including rule-based systems and classical machine learning models, require extensive manual effort for rule crafting and feature engineering (Elmagarmid et al., 2007). In recent years, deep learning (DL) methods have advanced the field by automatically learning representations of records, reducing the need for manual feature extraction and improving accuracy (Mudgal et al., 2018).

One of the most prominent approaches in modern EM is *Ditto* (Li et al., 2023), which leverages pretrained language models like BERT (Devlin et al., 2019) to generate embeddings for entity representations. *Ditto* has demonstrated state-of-the-art results in matching accuracy, particularly when dealing with complex datasets containing noisy data and scarce training examples. However, despite its effectiveness, *Ditto* suffers from high computational costs, making it less practical for large-scale or real-time applications. Additionally, it often requires fine-tuning on specific datasets, which can limit its generalizability.

In this paper, we propose EM-Join, a novel pipeline for entity matching that addresses both the accuracy and efficiency challenges. EM-Join leverages specialized language models to generate embeddings for concatenated attribute values, simplifying record representation. We then perform a similarity join using a heuristic method to select the best threshold, which is subsequently employed to determine whether two records represent the same entity. By reducing the complexity of the embedding generation process and optimizing the similarity join phase, EM-Join offers significant improvements in runtime without sacrificing accuracy.

We evaluated our method against *Ditto* using 13 publicly available datasets. Our method outperforms *Ditto* in 3 datasets while achieving comparable results in the remaining ones, with a notable reduction in execution time —up to 3 times faster than *Ditto*. Additionally, we compared our solution with *Ember* (Suri et al., 2022) and *DeepMatcher* (Mudgal et al., 2018), two other entity matching solutions. Our EM-Join method outperformed both *Ember* and *DeepMatcher* in accuracy across all datasets. These results demonstrate that our method provides a compelling trade-off between efficiency and effectiveness, making it a viable option for real-world applications where performance and speed are critical.

Santana, D. R., Lima, P. H. S. and Ribeiro, L. A.
EM-Join: Efficient Entity Matching Using Embedding-Based Similarity Join.
DOI: 10.5220/0013483700003929

In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 1*, pages 402-409
ISBN: 978-989-758-749-8; ISSN: 2184-4992

	Title	Author	Publisher
t_1	Pride and Prejudice	Jane Austen	Penguin Classics
t_2	Pride & Prejudice	Jane Austen	Penguin Classics
t_3	Pride and Prejudice	J. Austen	Penguin Classics

(a) Structured data.

	Description
t_1	Notebook Dell Inspiron 15, i7 11ª Geração, 16GB RAM, SSD 512GB, Windows 11
t_2	Dell Inspiron 15, Intel Core i7, 16GB RAM, 512GB SSD, Win 11 Home
t_3	Laptop Dell Inspiron 15, 11th Gen i7, 16GB, 512GB SSD, W11

(b) Textual data.

	Product	Company	Price
t_1	IPhone 16 Pro	Apple	999
t_2	Apple iPhone 16 Pro		999
t_3	iPhon 16Pro	Appl	999

(c) Dirty data.

Figure 1: Examples of datasets illustrating entity matching scenarios, including structured data, textual data, and dirty data. Each dataset contains records that can be considered matches, highlighting challenges such as structural differences, textual variations, and data imperfections.

2 BACKGROUND

2.1 Problem Definition

We follow the entity matching (EM) problem formalization used in (Mudgal et al., 2018). Given two data sources $\mathcal{A}$ and $\mathcal{B}$ with the same schema, each record represents a real-world entity. The objective is to identify the largest binary relation $\mathcal{M} \subseteq \mathcal{A} \times \mathcal{B}$, where each pair $(a,b) \in \mathcal{M}$ denotes that a and b refer to the same entity. If the task targets duplicate detection within a single dataset, we have $\mathcal{A} = \mathcal{B}$.

A labeled training dataset $\mathcal{T}$ is composed of tuples $\{(a^i, b^i, r)\}_{i=1}^{|\mathcal{T}|}$, where $\{(a^i, b^i)\}_{i=1}^{|\mathcal{T}|} \subseteq \mathcal{A} \times \mathcal{B}$ and r is a categorical label indicating whether a pair matches (*match*) or does not match (*no-match*). We use then $\mathcal{T}$ to train a classifier that categorizes pairs as "match" or "no-match. Figure 1 illustrates the EM challenges we address in this work, such as handling structured, textual, and dirty data.

2.2 Embeddings

Embeddings represent data as dense vectors, preserving semantic relationships. Early techniques, such as Word2Vec (Mikolov et al., 2013), capture word similarities but lack contextual adaptation. Advances in Transformer-based models (Vaswani et al., 2017) have enabled the creation of contextual embeddings. Sentence-BERT (Reimers and Gurevych, 2019), for instance, modifies the BERT architecture into a Siamese network structure to produce semantically meaningful, context-rich sentence representations, which is instrumental for tasks like EM.

2.3 Similarity Join

A similarity join identifies pairs of similar records from two datasets using a similarity function.

Definition 1 (Similarity Join)**.** *Given two sets of vectors, $\mathcal{V}_\mathcal{A}$ and $\mathcal{V}_\mathcal{B}$, and a threshold τ, the similarity join returns all pairs $\langle (a,b), s \rangle$ such that $sim(a,b) = s \geq \tau$.*

State-of-the-art techniques for similarity search on vector embeddings leverage proximity graph indexes to enhance efficiency. A prime example of such an index is the Hierarchical Navigable Small World (HNSW), which offers an excellent balance between speed and accuracy (Malkov and Yashunin, 2020). (Santana and Ribeiro, 2023) adapted HNSW's internal algorithms to optimize similarity join processing.

3 RELATED WORK

The EM problem, studied since the 1950s (Newcombe et al., 1959), has been addressed by communities like Databases, NLP, and Machine Learning, under terms like entity resolution, deduplication, and record matching. DeepMatcher (Mudgal et al., 2018), Ditto (Li et al., 2023), and Ember (Suri et al., 2022) are key DL-based solutions illustrating the evolution of EM methods. DeepMatcher uses flexible architectures with embeddings and attention mechanisms to process tuple pairs, outperforming previous learning-based techniques. Ditto fine-tunes pre-trained Transformers (BERT, RoBERTa) for EM tasks, supporting varying schemas and hierarchical data with high accuracy. (Lima et al., 2023) presented a comparative evaluation of DeepMatcher and Ditto on a wider range of textual patterns. Ember improves context enrichment in structured data through similarity joins, using Transformer-based embeddings to assemble fragmented data about entities.

EM solutions often rely on blocking techniques to reduce the quadratic complexity of the problem. Notably, the work in (Thirumuruganathan et al., 2021) defines a space of DL solutions for blocking. Other related problems in NLP and data integration,

like entity linking (Shen et al., 2015), entity alignment (Leone et al., 2022), and coreference resolution (Clark and Manning, 2016), often share interchangeable solutions. A review of pre-DL literature is in (Elmagarmid et al., 2007), and DL-based techniques are discussed in (Barlaug and Gulla, 2021).

4 EM-JOIN SOLUTION

EM-Join, our proposed solution to the EM problem, is structured into three stages: Preprocessing, Data Representation, and Join, as shown in Figure 2. Inspired by Ember, which focuses on data transformation and context enrichment, EM-Join is tailored to the EM problem, optimizing accuracy and efficiency in large-scale data scenarios.

In the first stage, **Preprocessing**, input data is loaded, and attributes from each record are concatenated into a single sentence, separated by the `<SEP>` token. This process creates a unified textual representation for each record in datasets $\mathcal{A}$ and $\mathcal{B}$, ensuring that all relevant attributes are captured and minimizing redundancy.

The second stage, **Data Representation**, involves transforming the concatenated records into embedding vectors using models from the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023). The selected model, loaded from Hugging Face[1], is fine-tuned to adapt to the dataset's characteristics. After fine-tuning, the model generates high-dimensional embeddings for each record in the datasets, resulting in sets of vectors $\mathcal{V}_\mathcal{A}$ and $\mathcal{V}_\mathcal{B}$. All vectors are further normalized to ensure consistency and comparability.

Algorithm 1: Join Step.

Input : Sets of vectors $\mathcal{V}_\mathcal{A}$, $\mathcal{V}_\mathcal{B}$; labeled training data $\mathcal{T}$; set of vectors $\mathcal{V}_{\mathcal{T}\mathcal{A}} = \{a | a \in \mathcal{V}_\mathcal{A}$ and a appears in $\mathcal{T}\}$; set of vectors $\mathcal{V}_{\mathcal{T}\mathcal{B}} = \{b | b \in \mathcal{V}_\mathcal{B}$ and b appears in $\mathcal{T}\}$; initial similarity threshold τ_0;
Output: Matching results $\mathcal{M}$

1 $\mathcal{S} \leftarrow SimJoin((\mathcal{V}_{\mathcal{T}\mathcal{A}}, \mathcal{V}_{\mathcal{T}\mathcal{B}}, \tau_0))$;
2 $\tau^* \leftarrow$ FindOptimalThreshold$(\mathcal{S}, \mathcal{T})$
3 $\mathcal{M} \leftarrow SimJoin((\mathcal{V}_\mathcal{A}, \mathcal{V}_\mathcal{B}, \tau^*))$
4 **return** $\mathcal{M}$

The final stage, **Join**, identifies record pairs in the input datasets that are considered matches, classifying pairs with a similarity score above a defined threshold. The optimal threshold is determined heuristically, as outlined in Algorithm 1.

Initially, a similarity join is performed on labeled subsets of $\mathcal{V}_\mathcal{A}$ and $\mathcal{V}_\mathcal{B}$ (Line 1) with a low starting threshold (e.g., 0.6) to ensure high recall. After optimizing the threshold using labeled data (Line 2), it is applied to the full datasets, retaining only pairs exceeding the threshold as matches (Line 3).

Algorithm 2: $SimJoin(\mathcal{V}_\mathcal{A}, \mathcal{V}_\mathcal{A}, \tau)$.

Input : Set of vectors $\mathcal{V}_\mathcal{A}$ and $\mathcal{V}_\mathcal{B}$; similarity threshold τ
Output: A set $\mathcal{M}$ containing all scored pairs $\langle (a,b), s \rangle$ s.t., $(a,b) \in \mathcal{V}_\mathcal{A} \times \mathcal{V}_\mathcal{B}$, and $sim(a,b) = s >= \tau$

1 $I \leftarrow BuildIndex(\mathcal{V}_\mathcal{A})$
2 **foreach** $b \in \mathcal{V}_\mathcal{B}$ **do**
3 $\mathcal{A} \leftarrow I.Search(b, \tau)$
4 **for** $a \in \mathcal{A}$ **do**
5 $s \leftarrow Sim(a,b)$
6 **if** $s >= \tau$ **then**
7 $\mathcal{M} \leftarrow \mathcal{M} \cup \langle (a,b), s \rangle$
8 **return** $\mathcal{M}$

Algorithm 3: FindOptimalThreshold.

Input : Sample Matching results $\mathcal{S}$, labeled training data $\mathcal{T}$
Output: Optimal threshold τ^*

1 Sort $\mathcal{S}$ by similarity score s in descending order
2 Initialize $F1^* \leftarrow 0$, $F1 \leftarrow 1$, $\tau \leftarrow \max(\mathcal{S}[s])$, and $\Delta\tau \leftarrow 0.05$
3 **while** $FI \geq F1^*$ **do**
4 $\mathcal{S}_\tau \leftarrow \{(a,b,s) \in \mathcal{S} \mid s \geq \tau\}$
5 $F1 \leftarrow$ ComputeF1$(\mathcal{S}_\tau, \mathcal{T})$
6 **if** $F1 > F1^*$ **then**
7 $F1^* \leftarrow F1$
8 $\tau^* \leftarrow \tau$
9 $\tau \leftarrow \tau - \Delta\tau$
10 $\mathcal{R} = \{\tau^* + k \cdot \delta\tau \mid k \in \{-4, -3, \ldots, 4\}, k \neq 0, \delta\tau = 0.01\}$
11 **for** $\tau \in \mathcal{R}$ **do**
12 $\mathcal{S}_\tau \leftarrow \{(a,b,s) \in \mathcal{S} \mid s \geq \tau\}$
13 $F1 \leftarrow$ ComputeF1$(\mathcal{S}_\tau, \mathcal{T})$
14 **if** $F1 > F1^*$ **then**
15 $F1^* \leftarrow F1$
16 $\tau^* \leftarrow \tau$
17 **return** τ^*

[1] https://huggingface.co

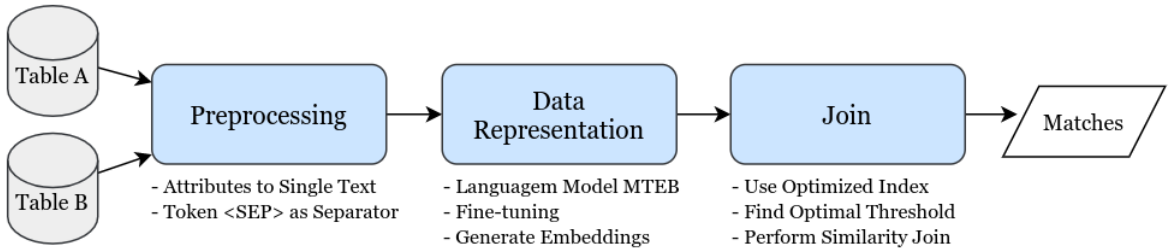

Figure 2: EM-Join architectural template.

Algorithm 2 describes the similarity join process, which evaluates cosine similarity for vector pairs $(a,b) \in \mathcal{V}_\mathcal{A} \times \mathcal{V}_\mathcal{B}$ and retains pairs satisfying $sim(a,b) \geq \tau$. To optimize efficiency, an HNSW index on $\mathcal{V}_\mathcal{A}$ is built (Line 1). Candidate pairs are formed by probing the index with the vectors in $\mathcal{V}_\mathcal{B}$ and those pairs meeting the similarity constraint are sent to the output (Lines 2–7).

The *FindOptimalThreshold* function (Algorithm 3) iteratively adjusts the threshold to maximize the F1-score. Initially, similarity scores are sorted (Line 1), and the threshold is reduced in decrements of $\Delta\tau$ until no further F1-score improvement is observed (Lines 3–9). A finer adjustment follows within a small range to determine the optimal threshold (Lines 11–16), which is then returned (Line 17).

EM-Join enhances precision and recall through heuristic-based threshold selection. However, it relies on labeled data, limiting its applicability in settings where such data is scarce. Alternative strategies are required for unsupervised threshold estimation.

Although F1-score is used for threshold selection, the method can be adapted to prioritize precision or recall based on specific requirements. For regulatory compliance or financial reconciliation, precision can be emphasized to ensure highly reliable matches. Conversely, for tasks like medical record linking or fraud detection, recall can be maximized to improve coverage. This adaptability makes EM-Join a versatile solution for various EM applications.

5 EXPERIMENTS AND RESULTS

This section presents an experimental study to evaluate the effectiveness of the EM-Join solution. EM-Join is compared to three established solutions: DeepMatcher, Ditto, and Ember. The evaluation is performed in two phases. First, we conduct an **Effectiveness Analysis** using 13 publicly available datasets to assess accuracy with the F1-score metric. Following that, we perform a **Runtime Evaluation**, comparing EM-Join exclusively to Ditto, which showed the best effectiveness results. The comparison highlights the strengths and limitations of EM-Join in terms of both accuracy and execution time. The EM-Join source code is available on GitHub[2].

5.1 Effectiveness Analysis

In this section, we evaluate the effectiveness of the proposed EM-Join solution using the F1-score metric, which provides a balanced measure of both precision and recall, making it particularly suitable for evaluating the performance of entity matching models in identifying duplicate records.

5.1.1 Datasets

We used 13 datasets from the DeepMatcher study (Mudgal et al., 2018), publicly available on GitHub[3], also employed in evaluations of Ember and Ditto. These datasets cover various domains, including products, publications, and businesses, with candidate pairs sampled from two tables with the same schema. The positive rate ranges from 9.4% to 25%, and the number of attributes per dataset ranges from 1 to 8. For consistency, we use the same 3:1:1 training, validation, and test splits. Table 1 summarizes the datasets, noting that some, like Abt-Buy and Company, are text-heavy, while others, like DBLP-ACM and iTunes-Amazon, contain noisy data. For comparison, we considered the results of the best-performing versions of DeepMatcher, Ditto, and Ember.

5.1.2 Experimental Setup

The EM-Join solution optimizes performance through specific parameters. In the *data representation* phase, two embedding models, *all-MiniLM-L12-v2* and *all-mpnet-base-v2*, with dimensions of 384 and 768, respectively, were selected for their efficiency and accuracy in semantic search (Muen-

[2]https://github.com/pauloh48/EM-Join

[3]https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

Table 1: Description of datasets.

Type	Dataset	Domain	Size	# Positives	# Attributes
Structured	Amazon-Google	software	11,460	1,167	3
	BeerAdvo-RateBeer	beer	450	68	4
	DBLP-ACM	citation	12,363	2,220	4
	DBLP-Scholar	citation	28,707	5,347	4
	Fodors-Zagats	restaurant	946	110	6
	iTunes-Amazon	music	539	132	8
	Walmart-Amazon	electronics	10,242	962	5
Textual	Abt-Buy	product	9,575	1,028	3
	Company	company	112,632	28,200	1
Dirty	iTunes-Amazon	music	539	132	8
	DBLP-ACM	citation	12,363	2,220	4
	DBLP-Scholar	citation	28,707	5,347	4
	Walmart-Amazon	electronics	10,242	962	5

Table 2: F1-scores of EM-Join compared to Ember (EMB), Deepmatcher (DM) and Ditto (DIT). Model 1 is all-MiniLM-L12-v2 and model 2 is all-mpnet-base-v2. Exact uses the IndexPlatIP index from the Faiss library that returns exact results, while HNSW is the index that returns approximate results. FT stands for Fine-tuning.

Type	Dataset	EMB (f1)	DM (f1)	DIT (f1)	EM-JOIN				
					Model	Exact FT on (f1)	HNSW FT on (f1)	Exact FT off (f1)	Best Threshold Found
Structured	Amazon-Google	70.43	69.3	75.58	1	76.03	76.03	47.78	0.71
					2	**78.06**	**78.06**	41.9	0.75
	BeerAdvo-RateBeer	91.58	72.7	**94.37**	1	90.32	90.32	86.67	0.85
					2	92.86	92.86	89.66	0.9
	DBLP-ACM	98.05	98.4	98.99	1	**99.32**	**99.32**	90.57	0.85
					2	98.43	98.43	86.47	0.85
	DBLP-Scholar	57.88	94.7	**95.6**	1	94.8	94.8	86.64	0.79
					2	93.75	93.65	81.63	0.77
	Fodors-Zagats	88.76	100	**100**	1	95.24	95.24	93.62	0.77
					2	95.45	95.45	89.36	0.81
	Itunes-Amazon	84.92	88.5	**97.06**	1	92.59	92.59	65.31	0.81
					2	94.34	94.34	83.33	0.85
	Walmart-Amazon	69.6	67.6	**86.76**	1	77.34	77.23	31.34	0.76
					2	77.39	77.09	31.25	0.82
Textual	Abt-Buy	85.05	62.8	**89.33**	1	82.76	82.76	33.22	0.76
					2	85.71	85.71	35.63	0.78
	Company	74.31	92.7	**93.85**	1	78.04	78.04	64.97	0.69
					2	90.21	90.21	73.33	0.67
Dirty	DBLP-ACM	97.58	98.1	99.03	1	**99.32**	**99.32**	89.97	0.86
					2	98.87	98.87	86.73	0.85
	DBLP-Scholar	58.08	93.8	**95.75**	1	94.6	94.6	86.01	0.77
					2	94.17	94.17	80.61	0.75
	Itunes-Amazon	64.65	79.4	**95.65**	1	89.66	89.66	61.22	0.8
					2	94.74	94.74	76.19	0.83
	Walmart-Amazon	67.43	53.8	**85.69**	**1**	77.78	77.47	29.17	0.77
					2	76.81	76.81	28.57	0.75

nighoff et al., 2023). Fine-tuning was done using the *sentence-transformers* library with fixed parameters: 40 epochs, batch size of 8, learning rate of 2×10^{-5}, and *ConstantLR* scheduler. In the *Join* phase, labeled data from the train and valid files were used to determine the optimal threshold, starting with 0.6. The *Faiss* library (Johnson et al., 2019) was used for similarity search, with *IndexFlatIP* for exact

matches and *HNSW* for approximate matches, configured with parameters $M = 64$, $efConstruction = 32$, and $efSearch = 32$. A heuristic approach was applied to handle the top-k limitation in *Faiss*. All experiments were conducted in Google Colaboratory using a GPU Nvidia Tesla T4 with 15 GB of memory.

5.1.3 Results

Table 3: Average F1-Score for each dataset type (Structured, Textual and Dirty) for EM-Join compared to Ember (EMB), Deepmatcher (DM) and Ditto (DIT).

Type	EMB	DM	DIT	EM-Join
Structured	80.17	84.46	92.62	90.35
Textual	79.68	77.75	91.59	87.96
Dirty	71.94	81.28	94.03	91.61
Average	77.26	81.16	92.75	89.97

Table 2 summarizes EM-Join's effectiveness compared with the competitors, showing F1-scores for different configurations, including *IndexFlatIP* and *HNSW* indexes with and without fine-tuning. EM-Join performed competitively, outperforming Ditto in three datasets (Amazon-Google, structured; and DBLP-ACM, both structured and dirty), achieving higher F1-scores in cases such as Amazon-Google (78.06 vs. 75.58) and DBLP-ACM structured (99.32 vs. 98.99). No significant differences were observed between the exact and approximate HNSW indexes, except in DBLP-Scholar and Walmart-Amazon, where exact fine-tuning marginally outperformed HNSW. While Ditto had higher scores in some datasets, such as DBLP-Scholar and Fodors-Zagats, EM-Join's performance was dataset-dependent, with fine-tuning showing a marked improvement, as seen in Amazon-Google where disabling fine-tuning resulted in lower F1-scores (47.78 and 41.90).

Table 3 presents the average F1-Scores for all approaches across Structured, Textual, and Dirty datasets. Ditto achieved the highest average F1-Score (92.75), followed by EM-Join (89.97). EM-Join performed strongest on Structured datasets (90.35), close to Ditto (92.62), but showed a gap on Textual datasets with an F1-Score of 87.96, compared to Ditto's 91.59. On Dirty datasets, EM-Join achieved 91.61, performing worse than Ditto (94.03) but outperforming Ember (71.94) and DeepMatcher (81.28). These results highlight EM-Join's strengths in Structured and Dirty datasets, with potential for improvement on Textual datasets, particularly for unstructured data.

Table 4 details the execution time for various steps in the EM-Join process, comparing fine-tuning, embedding generation, and similarity joins with the exact (*IndexFlatIP*) and approximate (HNSW) indices. The analysis includes two embedding models: *all-MiniLM-L12-v2* (Model 1) and *all-mpnet-base-v2* (Model 2). The HNSW index significantly reduced execution time, with the *Itunes-Amazon* dataset (Model 2) showing a 78.7% reduction in time (83s vs. 390.6s) compared to the exact join. Similarly, on *Walmart-Amazon* (Model 1), the time decreased by 51.9% (27.1s vs. 56.4s). Despite these time savings, F1-Scores remained largely unchanged, demonstrating that HNSW improves efficiency without compromising matching quality.

Training times also differ between models, with Model 2 taking longer due to its larger size and higher dimensionality. While Model 2 generally provides higher F1-scores, Model 1 outperformed Model 2 on datasets like *DBLP-ACM*, *DBLP-Scholar*, and *Fodors-Zagats*, suggesting a trade-off between embedding quality and training time, particularly in resource-constrained environments.

5.2 Runtime Evaluation

In this section, we evaluate the computational performance of the proposed EM-Join solution by comparing its execution time with that of Ditto.

5.2.1 Datasets

The datasets used were reduced versions of Big-Citations and Song-Song from Das et al. (Das et al., 2017). Big-Citations originally contained two tables and a gold standard with over half a million pairs, while Song-Song had over one million pairs. A three-step reduction technique was applied: first, 10% of the gold standard pairs were randomly sampled; then, the necessary records from the original tables were identified and proportionally reduced; and finally, the gold standard was updated to match the reduced tables. The final datasets contained 10–21% of the original table sizes. A combined dataset, including negative pairs and the reduced gold standard, was split into training, validation, and test sets using `scikit-learn`, ensuring balanced partitions.

5.2.2 Experimental Setup

The experiments used EM-Join and Ditto. For EM-Join, the *all-mpnet-base-v2* model was employed with the parameters defined in Section 5.1.2. Ditto used the RoBERTa model with a batch size of 8, a maximum input length of 256 tokens, a learning rate of 2e-5, and 40 epochs for fine-tuning. Data augmentation, entry swapping, attribute deletion, model checkpointing, and mixed precision (FP16) training were applied. The experiments ran on a Supermicro AMD compute node with 192 cores, 768 GB of RAM, 1

Table 4: Time spent in the EM-Join execution steps: fine-tuning (FT), generation of embeddings (ENC), performing the similarity join with the exact index and for build and joining with the HNSW index. Model 1 is all-MiniLM-L12-v2 and model 2 is all-mpnet-base-v2.

Type	Dataset	Model	FT (s)	ENC (s)	Join Exact (s)	Build HNSW (s)	Join HNSW (s)
Structured	Amazon-Google	1	642	3.6	2.5	1.2	0.7
		2	996	10.0	4.9	2.4	1.2
	BeerAdvo-RateBeer	1	37	5.2	6.7	2.6	2.3
		2	67	17.3	14.5	5.0	4.0
	DBLP-ACM	1	717	4.8	3.8	0.6	0.7
		2	1657	17.1	6.3	1.0	1.3
	DBLP-Scholar	1	1576	60.5	85.4	68.0	2.5
		2	3596	201.7	172.1	128.0	4.3
	Fodors-Zagats	1	68	0.9	0.2	0.1	0.1
		2	135	2.8	0.3	0.2	0.2
	Itunes-Amazon	1	44	73.9	211.7	40.1	4.2
		2	119	285.9	390.6	74.9	8.1
	Walmart-Amazon	1	682	25.0	29.0	13.5	2.3
		2	1427	86.9	56.4	23.9	3.2
Textual	Abt-Buy	1	644	3.1	1.2	0.3	0.4
		2	2286	9.5	2.1	0.5	0.8
	Company	1	7440	316.5	393.8	518.5	376.7
		2	23220	1434.6	827.2	683.6	371.4
Dirty	DBLP-ACM	1	781	4.9	5.3	0.6	0.7
		2	1814	17.6	6.5	1.2	1.4
	DBLP-Scholar	1	1762	60.7	84.0	76.4	2.6
		2	3451	206.3	171.5	121.7	4.4
	Itunes-Amazon	1	55	76.1	198.9	45.3	5.3
		2	123	282.6	427.2	86.9	9.7
	Walmart-Amazon	1	571	24.7	34.7	15.0	2.0
		2	1408	80.3	64.3	26.4	3.3

Table 5: Execution times (in seconds) for Ditto and EM-Join on the Big-Citations and Songs datasets. The table details the total runtime for Ditto and the breakdown of EM-Join's runtime into its main stages: fine-tuning, encoding, automatic threshold calculation, and join operation.

Dataset	Ditto	EM-Join				
	Total	Fine-tuning	Encodding	Auto Threshold	Join	Total
Big Citations	**19649**	5469	675	536	199	**6879**
Songs	**49080**	12755	549	676	107	**14087**

TB of storage, and three NVIDIA A100 GPUs with 80 GB of memory, using Conda to create an isolated virtual environment.

5.2.3 Results

Table 5 shows the execution times for each stage of EM-Join and the total execution time for both approaches. For EM-Join, the runtime is divided into four stages: fine-tuning, encoding, automatic threshold calculation, and join operation. On the Big-Citations dataset, EM-Join achieved a total runtime of 6879 seconds, reducing the execution time by approximately 2.8 times compared to Ditto, which required 19649 seconds. On the Songs dataset, EM-Join completed in 14087 seconds, achieving a reduction of over 3.4 times compared to Ditto's 49080 seconds.

The results demonstrate the efficiency of EM-Join, particularly in its modular structure, which allows each stage to be optimized independently. Fine-tuning was the most computationally intensive step, accounting for the largest portion of the runtime. Despite this, EM-Join consistently achieved substantial runtime reductions, showcasing its scalability and effectiveness for large-scale entity matching tasks, with improvements of up to 3.4 times over Ditto while maintaining similar levels of result quality.

6 CONCLUSION

This paper proposed a new EM technique that combines text embeddings generated by pre-trained language models with a similarity join mechanism. By optimizing the matching process through heuristic threshold selection, our method achieved competitive accuracy, outperforming the accuracy of *Ditto*, the state-of-the-art EM solution, in 3 of the 13 tested datasets, while significantly reducing execution time — up to 3 times faster than *Ditto*. These results demonstrate the effectiveness of our approach in balancing performance and speed, making it suitable for large-scale, real-time applications.

For future work, we plan to refine the threshold selection process to further improve accuracy, particularly on textual and dirty datasets. We also intend to explore the applicability of our method in other application domains and larger datasets. Additionally, integrating more advanced language models and optimizing computational efficiency will be key areas of focus to expand the versatility, robustness, and scalability of our proposed solution.

ACKNOWLEDGEMENTS

This work was partially supported by CAPES/Brazil and LaMCAD/UFG.

REFERENCES

Barlaug, N. and Gulla, J. A. (2021). Neural Networks for Entity Matching: A Survey. *ACM Transactions on Knowledge Discovery from Data*, 15(3):52:1–52:37.

Clark, K. and Manning, C. D. (2016). Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In *Proceedings of the Association for Computational Linguistics*, pages 643–653.

Das, S., G.C., P. S., Doan, A., Naughton, J. F., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V., and Park, Y. (2017). Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. SIGMOD '17, page 1431–1446, New York, NY, USA. ACM.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the ACL*, pages 4171–4186.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

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

Leone, M., Huber, S., Arora, A., García-Durán, A., and West, R. (2022). A Critical Re-evaluation of Neural Methods for Entity Alignment. *Proceedings of the VLDB Endowment*, 15(8):1712–1725.

Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W.-C. (2023). Effective entity matching with transformers. *The VLDB Journal*, 32(6):1215–1235.

Lima, P. H. S., Santana, D. R., Martins, W. S., and Ribeiro, L. A. (2023). Evaluation of Deep Learning Techniques for Entity Matching. In *International Conference on Enterprise Information Systems*, pages 247–254.

Malkov, Y. A. and Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. In Bengio, Y. and LeCun, Y., editors, *International Conference on Learning Representations*.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the SIGMOD Conference*, pages 19–34. ACM.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2023). MTEB: Massive Text Embedding Benchmark. In *Proceedings of the ACL*, pages 2014–2037, Dubrovnik, Croatia. ACL.

Newcombe, H., Kennedy, J., Axford, S., and James, A. (1959). Automatic Linkage of Vital Records. *Science*, 130(3381):954–959.

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992.

Santana, D. R. and Ribeiro, L. A. (2023). Approximate Similarity Joins over Dense Vector Embeddings. In *Proceedings of the Brazilian Symposium on Databases*, pages 51–62. SBC.

Shen, W., Wang, J., and Han, J. (2015). Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Suri, R., Fischer, J., Madden, S., and Stonebraker, M. (2022). Ember: No-code context enrichment via similarity-based keyless joins. *Proceedings of the VLDB Endowment*, 15:699–712.

Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., and Doan, A. (2021). Deep Learning for Blocking in Entity Matching: A Design Space Exploration. *Proceedings of the VLDB Endowment*, 14(11):2459–2472.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 5998–6008.