Entity Resolution for Streaming Data with Embeddings

Zhongwei Ma, Philippe Roose, Jiefu Song

Entity Resolution for Streaming Data with Embeddings

Zhongwei MA[1,2][0009−0009−4276−1821], Philippe ROOSE[2][0000−0002−2227−3283], and Jiefu SONG[3][0000−0002−2066−7051]

[1] Technopôle Domolandes, Saint Geours de Maremne, France
[2] Universite de Pau et des Pays de l'Adour, E2S UPPA, LIUPPA, Anglet, France
{zhongwei.ma, philippe.roose@univ-pau.fr}
[3] Institut de Recherche en Informatique de Toulouse - Université Toulouse Capitole, 31000, Toulouse, France jiefu.song@ut-capitole.fr

Abstract. Most data streaming systems collect and process real-time data from various sources. However, many decisions require both streaming and historical data to obtain a comprehensive view. Entity resolution (ER) determines whether two records refer to the same real-world entity in the absence of a shared identifier. Therefore, it is crucial to integrate stream data and stored data together through entity resolution and to ensure the accessibility and usability of data from different sources. Existing ER methods often fail to support incremental stream processing while efficiently handling complex data like text. Given the strength of embedding techniques in capturing semantic and syntactic information, we aim to adapt embedding-based ER to streaming data for integrating incoming and existing records. We propose a dynamic graph embedding suitable for stream processing to perform entity resolution within relational tables.

Keywords: entity resolution · data stream · embedding

1 Introduction

Entity resolution, also called record linkage or data matching, aims to find different descriptions that refer to the same entity in the real world appearing either within or across data sources, when there are no unique entity identifiers, ensuring the accuracy and consistency of integrated datasets [10]. Assuming that we have real-time sales records across different retailers' platforms, if these records are integrated with historical records in our database, we can gain deeper insights into popular products and consumer purchasing trends. This is a crucial step when we perform data integration and want to offer a uniform view of autonomous and heterogeneous data sources, facilitating further processing [20].

In the big data era, streaming data—continuous and real-time—is increasingly common [3], originating from sources like social media and IoT devices.

This is the author's version of the work. The final authenticated version is available online at: https://doi.org/10.1007/978-3-032-02215-8_8

The requirements for data stream analytics come from two aspects: real-time processing to support immediate decision-making with low latency, and storage of data for subsequent queries or more complex and time-consuming analysis. For instance, in the IoT industry, millions of devices generate continuous data streams and report abnormal or significant events in real time, optimizing online production. By integrating and analyzing this data with historical consumer trends, companies can unlock greater value, enhance consumer insights, and maximize profitability. Therefore, modern data stream processing systems need to incorporate both stream and batch processing [17] within a unified framework, as outlined in the Lambda architecture originally introduced by Matz and Werren [22]. To enhance the efficiency of these systems, integrating stream data with historical data by entity resolution is essential.

The main challenge of stream data lies in its unbounded nature, which requires a dynamic framework capable of performing entity resolution incrementally. The lack of a complete dataset also makes it complicated because we can't determine similarity between two descriptions and identify the best match within a finite amount of data.

ER has been defined historically as an offline task performed during data integration to improve data quality in databases [10], and many attempts have been made to focus on batch processing using different methods [13, 9, 19, 25, 26]. However, these batch-oriented approaches designed for static databases are not well suited for streams. So far, two main approaches have been adapted for incremental data. The first is rule-based methods [15, 30]. While effective in constrained domains, these approaches often rely on complex rules and domain-specific knowledge, which limits their generalizability. The second approach involves machine learning, particularly clustering techniques [11, 31]. However, this approach faces challenges when applied to unstructured textual data, limiting its effectiveness in real-world environments.

To address the limitations in text analysis and the complexity of domain-specific algorithms, we turn our attention to embedding techniques to map text into low-dimensional vectors capturing semantic relationships [34]. For now, this technique is usually used for batch processing [19]. Additionally, significant research has been conducted on incremental embeddings [4], laying the foundation of entity resolution in streaming data.

In this work, we introduce a framework for data integration of streaming data by dynamic graph embedding. This framework includes a complete process from pre-trained embedding models by local datasets to accepting streaming data and gradually involves the added data and determining whether the added data belongs to any entity present in the original dataset. The construction of embedding represents well the semantic and syntactic information [34].

Our contributions is a dynamic graph embedding model for streaming entity resolution. Building on embedding techniques that capture relationships and semantic information, our model incrementally updates both the graph and its embeddings. Specifically, when a new record arrives, we add a corresponding node, create edges connecting it to relevant parts of the existing graph, and generate

evolving random walks from the new node to assess similarity. This incremental approach avoids retraining from scratch, significantly reducing computational costs while enabling efficient and scalable resolution of entities in data streams.

The article is structured as follows. In section 2, we list common methods used for Entity Resolution and analyze their characteristics. Section 3 introduces the conception of our proposal. In section 4, we describe in detail the implementation of the protocol and present an analysis of the results. We conclude in the last section and outline directions for future work.

2 Related work

We review the literature related to this paper in three areas. Section 2.1 summarizes the methods suitable for streaming data in order to present the current situation of incremental entity resolution. Section 2.2 focuses on embedding techniques and illustrates how they can be applied to achieve entity resolution. Finally, section 2.3 explores dynamic embeddings, detailing how existing tools can adapt embedding methods in batch for real-time stream data processing.

2.1 Entity resolution for stream

Many studies mentioned in [5] have explored various approaches for entity resolution in order to face the challenge of incremental data. We categorize them into three main groups: Query-driven approaches, Rule-based approaches, and Learning-based approaches.

Query-driven approaches [32, 2] store all incoming data and perform entity resolution at query time. However, these on-demand methods can take several minutes to wait for a result. Additionally, redundant records further increase storage demands, adding to resource inefficiency.

Rule-based approaches [15, 30, 14] are one of the most commonly used methods in entity resolution, which rely on deterministic and interpretable rules to compare record attributes. However, the realization of this approach can be very complex and requires sophisticated algorithms designed for streaming data.

Learning-based approaches have become increasingly prevalent due to advancements in machine learning techniques. Most of these approaches operate in batch mode [26] and aim to enhance accuracy and efficiency through methods such as graph matching and neural networks. However, training machine learning models presents significant challenges, particularly with noisy data and limited labeled resources, especially in streaming scenarios, where datasets are often incomplete. Incremental methods often use unsupervised techniques like clustering [31]. Instead of linking records individually, the objective of clustering is to group records into their true and often latent entities. Despite their advantages, the performance of clustering methods is limited when processing high-dimensional data, such as text, which is crucial in entity resolution tasks. Moreover, dirty data, such as spelling mistakes or data with missing values, can cause noise and false

positives, which amplifies inaccuracies, especially in the context of continuous data streams [10].

Therefore, existing propositions of entity resolution for streaming satisfy some requirements in real-time situations, like incremental update. But they do not have excellent performance as for large-scale and complex data streams such as unstructured textual data. This kind of data can be analyzed by embedding models efficiently, which we will present in the next section.

2.2 Embeddings and entity resolution

Word embeddings are a key technique in machine learning, mapping high-dimensional data into fixed-length vectors. This representation captures semantic relationships between words, ensuring that similar words have similar vector representations [1].

Various methods can be used to obtain this type of representation. Prediction-based approaches (e.g., word2vec [23]) train neural networks by leveraging local data (e.g., a word's context) to either predict a word based on its context or infer the context from a given word. This process ensures that words with similar semantic meanings are mapped to similar vector representations. Count-based methods (e.g., GloVe [28]), on the other hand, build co-occurrence matrices to extract global statistical patterns, such as corpus-wide word frequencies, then reduce dimensions to obtain semantically meaningful vector representations. These powerful semantic understanding capabilities significantly improve the analysis of dirty data and reduce errors. Below are key applications of embedding techniques in the domain of entity resolution.

DeepER [12] was one of the first methods to use word embeddings (e.g. GloVe). It applies Long Short-Term Memory (LSTM) networks to learn relationships between attributes in the tuple by labeling data and converting them into a vector representation of fixed dimensions. Then similarity features are calculated and fed into a binary classifier to determine entity matches. DeepMatch [25] follows a similar approach but with more choices for attribute embedding and similarity representation. They demonstrate competitive performance when handling the datasets containing a moderate level of dirty data.

Ditto [19] uses pre-trained transformer-based models for feature extraction and fine-tunes them as a binary classifier for entity matching. It handles dirty data better through a structured serialization process and data augmentation techniques. [7] also evaluate transformer-based models and explore attention mechanisms for the entity resolution task. However, there are significant energy and computational costs associated with using large language models (LLM).

More recently, several frameworks have represented data or record pairs using embedding-based approaches such as multiEM [35], Unicorn [33] and FlexER [16]. These frameworks employ advanced techniques, including graph neural networks, to identify record similarities and enhance generalization capabilities. Nevertheless, they encode data as fixed, linear sequences based on predefined rules, which restricts their ability to interpret relational tables, for example, column relationships. To achieve a more comprehensive interpretation, Embdi

[8] introduces graphical representations into entity resolution tasks to enrich the information captured by embeddings.

Despite these advancements, all the mentioned methods operate in batch mode, which limits their application to real-time scenarios. In order to deal with continuously arriving data, an executable method is required to incorporate the data progressively without manually restarting the program each time.

2.3 Dynamic embeddings

Traditionally, word embeddings are trained in batch mode. To support incremental learning, several models have extended classical algorithms. Incremental Skip-Gram (ISG) [18] adapts Word2Vec to streaming data by updating word frequencies and noise distributions using the Misra-Gries algorithm [24] for fixed vocabulary management. The incremental GloVe model [27] reformulates the original loss function recursively, allowing continuous updates without retraining on the full dataset. The incremental word matrix approach [6] maintains a word-context matrix by replacing infrequent or outdated entries when memory is limited.

As far as the entity resolution problem is concerned, in addition to dealing with plain text itself, we can construct a graph to deal with the problem, such as Embdi. In contrast to limited evolution in text (e.g., adding words), graph evolution can take many forms, such as the addition of nodes or edges, the evolution of edge weights, and more. The field of dynamic graph embedding is well-researched, with various methods [4] depending on how graph embeddings are constructed and how the graph evolves over time. For instance, dynnode2vec [21] utilizes the dynamic Skip-Gram model to generate embedding vectors. It leverages the pre-trained Skip-Gram model as initial weights and updates the vocabulary based on new evolving walks, instead of retraining the entire graph.

Therefore, although most embedding methods for entity resolution still operate in batch mode, incremental language models suggest that existing batch processing techniques can be adapted for streaming processing.

In summary, while existing methods for incremental entity resolution provide valuable experience, they still lack generalizability across diverse data sources and scalability for continuous data streams. Besides, ensuring stability when handling dirty data remains a challenge. Embedding techniques offer a universal solution for capturing text semantics, with dynamic embeddings showing strong potential for streaming applications. Additionally, we propose a dynamic embedding-based approach for entity resolution tasks, detailed in the next section.

3 Proposal

3.1 Problem statement

In this section, we clarify the problem and introduce key definitions that guide our approach.

To simulate the integration of streaming data into a local dataset, we assume two data sources: one is a static dataset in which each record corresponds to a

different entity, and another consists of incoming streaming data. The goal of incremental entity resolution is to match an arriving record to its most similar entity in an evolving reference dataset. This reference dataset includes both the static dataset and previously observed streaming records, dynamically updating over time as new data arrives.

To formalize the problem, we introduce some main symbols and concepts used in incremental approaches in Table 1.

Table 1: Symbol Definitions

Symbol	Explanation
D	A static dataset, which can be empty
$\mathcal{D}$	A dataset of streaming data
ΔD	The increment of incremental dataset over a period of time
R_i	An arbitrary record
$\mathcal{C}_t$	A set of all processed data prior to time t, which serves as the input for the next round of processing
M_{t,R_i}	The best match for record R_i at time t
$\mathcal{M}_t$	The match list for all records at time t
$sim(\cdot,\cdot)$	A similarity function for matching records

Entity resolution in data streaming presents two main problems.

The first is from an incomplete dataset for streaming. At time t, we only have a partial view of $\mathcal{D}$, making it impossible to wait until the full dataset is available before performing the process. More specifically, we can represent the dataset $\mathcal{D}$ at one moment as follows,

$$\mathcal{D}_n = \bigsqcup_{i=1}^{n} \Delta D_i \tag{1}$$

$\mathcal{D}_n$ represents a dataset formed by combining data increments $\Delta D_1, \ldots, \Delta D_n$, where each increment ΔD_i corresponds to new data arriving within a specific time interval $[t_{i-1}, t_i]$. We assume that these intervals are sequential and non-overlapping, meaning that each new time interval meets the previous one, expressed by $[t_{i-1}, t_i]$ meets $[t_i, t_{i+1}]$. Thus, the overall dataset $\mathcal{D}_n$ evolves over time by sequentially adding these increments.

Instead of comparing the whole dataset in batch mode, which is impossible because the data stream is generated continuously with no end point, the process should operate incrementally. A reasonable way is the execution of the entity resolution process each time ΔD_n arrives in order to not accumulate too much data. The timing of these increments can be determined by various factors, such as a predefined number of newly arrived records or fixed time intervals.

The main idea of incremental entity resolution is reusing the results of previous processing to avoid repetitive calculation. In this case, for incremental data ΔD_n et time t, the previously processed data is stored in a candidates set, represented by $\mathcal{C}_t = D \cup \mathcal{D}_{n-1}$. These data have been transformed from their original state to a form ready for comparison. At each incremental update, the candidate set is continuously integrated with new data, while the previously processed data

remains unchanged and is only used for comparison in the similarity function. Fig. 1 shows the data involved in this incremental process over time.

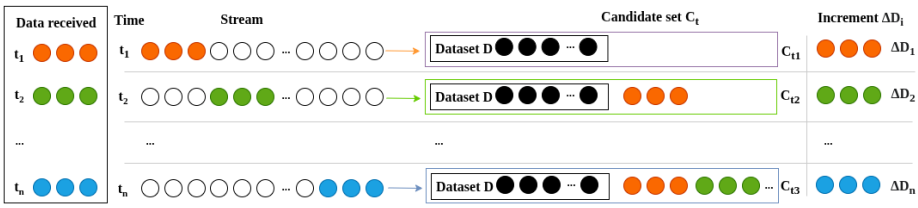


Fig. 1: Data involved in incremental process over time

So we can define the first problem as follows:

Definition 1. *At any time t, incrementally update the candidate set of $\mathcal{C}_t$ and compare data pairs by similarity function $sim(\Delta D_n, \mathcal{C}_t)$ to determine M_{t,R_i} for each element $R_i \in \Delta D_n$.*

The second problem is that matching is not a static process over time. Since candidate records change over time because of the continuous nature of the stream, previously resolved matches M_{t',R_i} (where $t' < t$) may need to be updated when new records arrive. This requires continuously maintaining consistency in match results. We define the match set at time t as $\mathcal{M}_t = \{(R_i, M_{i,R_i}) \mid i \leq t\}$. With growth of $\mathcal{M}_t$, storing all historical matches becomes infeasible. To ensure efficiency, we need a strategy to determine when and which matches should be retained or discarded. We resume as the following:

Definition 2. *Handle and maintain the dynamic matching results over time $\mathcal{M}_t$, such that at any time t, the correct corresponding similar result is included in the list once it appears, and its similarity value remains relatively high compared to all potentially similar records in the list.*

Our proposed pipelines address these problems through an incremental resolution process that efficiently updates matches while minimizing storage overhead. The key feature of our mechanism is to maintain all relevant matching information in a single index, enabling uniform query access. The details of our solution are presented in the following section.

3.2 Proposition

Our framework[3] for incremental entity resolution consists of three main parts: data preparation, incremental embedding construction, and matching list construction. In this work, we reuse the approach for building an embedding model in

[3] https://github.com/Mzhongwei/er_embedding_streaming

[8], which is suitable for extension for streaming data and represents rich semantic information in a lightweight language model. However, instead of constructing an embedding model based on two complete datasets at once, we introduce incremental embeddings that update the model over time to improve the scalability of the streaming application. To illustrate how these three parts fit together and the flow of data between them, we summarize the process as follows.

Data preparation Data preparation processes raw data increments ΔD_i from data sources. Metadata extraction is conducted at this stage to provide concise and representative descriptions of real-world entities. For instance, in the case of scientific articles, metadata may include attributes such as the author's name, title, and publication date. A relational table is then constructed based on the extracted metadata, providing a flexible foundation for various tasks across different domains. In this work, it is specifically used for entity resolution. This data preparation step transforms unstructured data into a structured format, effectively reducing data volume and standardizing it for downstream processing.

Incremental embedding construction Based on this relational table of data increments, we construct an incremental embedding model. The model used in this process is pre-trained on a local dataset D, but pre-training can be ignored in case we integrate data streams from scratch. The training pipeline is structured as follows: each record in the relational table is first assigned a unique identifier. Every value, including ID numbers and column names, is treated as a node in an undirected heterogeneous graph. A random walk is then performed on this graph to generate contextual representations of these values and tokens in the form of sentences. These sentences are subsequently mapped into a fixed-length vector space, which forms the embedding representation for values [8].

Algorithm 1: Incremental Embedding Algorithm

Input: data increment ΔD_i, embedding model $\mathcal{E}_{i-1}$, graph $\mathcal{G}_{i-1}$
Output: updated embedding model $\mathcal{E}_i$, updated graph $\mathcal{G}_i$

Initialize evolving nodes list $\mathcal{L}$
Initialize graph $\mathcal{G}_i \leftarrow \mathcal{G}_{i-1}$
foreach $R \in \Delta D_i$ **do**
    $\mathcal{G}_i, \Delta N_1 \leftarrow \text{addNodesToGraph}(\mathcal{G}_i, R)$
    $\mathcal{G}_i, \Delta N_2 \leftarrow \text{addEdgesToGraph}(\mathcal{G}_i, R)$
    $\mathcal{L} \leftarrow \text{addEvolingNodeToList}(\mathcal{G}_i, \Delta N_1, \Delta N_2)$
Generate new random walks for $\mathcal{L}$: $\mathcal{W} = \text{GenerateRandomWalk}(\mathcal{G}, \mathcal{L})$;
Train model with walks $\mathcal{W}$: $\mathcal{E}_i = \text{UpdateEmbedding}(\mathcal{E}_{i-1}, \mathcal{W})$
return $\mathcal{E}_i$, , $\mathcal{G}_i$

The incremental pipeline follows the same structure but incorporates new data dynamically. The whole process is shown in algorithm 1. When a data increment ΔD_i arrives at time t_i, we take the graph $\mathcal{G}_{i-1}$ and the embedding model $\mathcal{E}_{i-1}$ at time t_{i-1} as initial variables. The graph is updated by adding new nodes with values extracted from each record R of data incremental ΔD_i

if the corresponding values do not already exist; then new edges are added to the graph in order to represent the relation with different records and different values within one record (cf. 3-5). In this step, all the nodes that have evolved, both new nodes ΔN_1 as well as nodes with changed edges ΔN_2, are recorded in an evolving nodes list $\mathcal{L}$ (cf. line 6). New random walk $\mathcal{W}$ then starts from nodes in this list and generates new sentences (cf. line 7). These sentences are used to refine the embedding model through incremental learning, ensuring that the representations remain up to date without retraining from the beginning (cf. line 8). The updated graph $\mathcal{G}_i$ and embedding model $\mathcal{L}_i$ are stored for the next iteration.

Matching list construction Matching list construction takes as input ID numbers of incremental records to represent these records and the embedding model updated by the previous step. As shown in algorithm 2, for each record represented by one identifier, we can calculate similarity degrees between vectors of the ID numbers to identify the most similar records according to the vector representations in the embedding model (cf. line 2). Then we select the top k most similar matches (cf. line 3), store their relationships and their degree of similarity symmetrically in a list, where we retain only n most similar matches for each record and remove the matches sorted after n (cf. line 4-6).

Algorithm 2: Matching List Construction

Input: Incremental record IDs $\mathcal{ID} = \{id_1, id_2, \ldots, id_n\}$, embedding model $\mathcal{E}_i$, candidate set $\mathcal{C}$, number of most similar records k after calculation and n in matching list, previous matching list $\mathcal{M}_{i-1}$

Output: Updated matching list $\mathcal{M}_i$

1: **foreach** $id \in \mathcal{ID}$ **do**
2: Execute similarity function: Sim($\mathcal{E}_i$, id)
3: Select top-k most similar IDs with their similarity s: $\mathcal{N}_{id} = \{(j_1, s_1), (j_2, s_2), \ldots, (j_k, s_k) \mid j \in \mathcal{C}\}$
4: $\mathcal{M}_i[id] \leftarrow$ top-n most similar elements of $\mathcal{M}_{i-1}[id] \cup \mathcal{N}_{id}$;
5: **foreach** $(j, s) \in \mathcal{N}_{id}$ **do**
6: $\mathcal{M}_i[j] \leftarrow$ top-n most similar elements of $\mathcal{M}_{i-1}[j] \cup \{(id, s)\}$;
7: **return** $\mathcal{M}_i$

For example, given an ID number 001, we compute its most similar ID 002, 003, 004 with similarity representatively 0.8, 0.9, 0.7. To ensure bidirectional consistency, we record both the forward and reverse association, such as $001 \rightarrow (002, 0.8), (003, 0.9), (004, 0.7)$, $002 \rightarrow (001, 0.8)$, $003 \rightarrow (001, 0.9)$, $004 \rightarrow (001, 0.7)$. Consider also that we keep only the 2 most similar values for each record, so the final result is like $001 \rightarrow (002, 0.8), (003, 0.9)$, $002 \rightarrow (001, 0.8)$, $003 \rightarrow (001, 0.9)$, $004 \rightarrow (001, 0.7)$. This list dynamically evolves as new comparisons are performed over time.

4 Experiments

4.1 Protocol

In this section, we present an experimental evaluation of our proposition. The main objective of our experiments is to evaluate the performance of our proposed real-time processing architecture and its capacity to handle incremental data. Our evaluation aims to answer the following research questions:

RQ1. What are the effectiveness and performance of our proposition ?

RQ2. How does the size of the incremental training data affect the experimental results of incremental learning?

Datasets We use three open-source datasets[4] created from real-world data and are dedicated to the entity resolution task. Their information is described in Table 2.

Table 2: Overview of datasets used in the evaluation.

Name	**Columns**	**Tuples - Source A**	**Tuples - Source B**	**Matches**	**Description**
Amazon	Id, title, price, manufacturer	1363	3226	1167	Commodity list from Amazon
DBLP-ACM	Id, title, venue, year, author_1, author_2, author_3, author_4	2294	2617	2224	Scientific articles
Fordors-zagats	Id, name, addr, city, phone, type, class	533	331	110	Personal information

Evaluation metrics We use precision, recall and F1-score as primary metrics for performance. Since it is difficult to define the end of the task during stream processing, we calculate metrics based on the overall results after all Source B data have been processed. We sort the list in descending order of similarity and select $x (x \leq \mathit{listLength})$ records with similarity ranked in the top k. These records are then grouped into candidate pairs. After removing duplicates, the remaining predicted pairs are compared with the ground truth, where records refer to the same entity. The number of correctly predicted pairs corresponds to the intersection between the predicted and ground truth pairs.

[4] https://zenodo.org/records/7930461

Implementation To simulate the data streaming application, we take the tuples in Data Source A as the local dataset used for pre-training, and send the data from source B to kafka producer incrementally as a data stream at a regular and fixed frequency without concurrency. For each record of source B, we look for its similar values from Source A and construct a sequence of up to 10 in both directions (as mentioned in Section 3.2) to analyze how well the similarity-based ranking aligns with the correct pairing provided by the ground truth. In this article, we directly utilize relational data. Therefore, the Data Preparation will not be discussed here. The whole process is illustrated in Fig. 2.

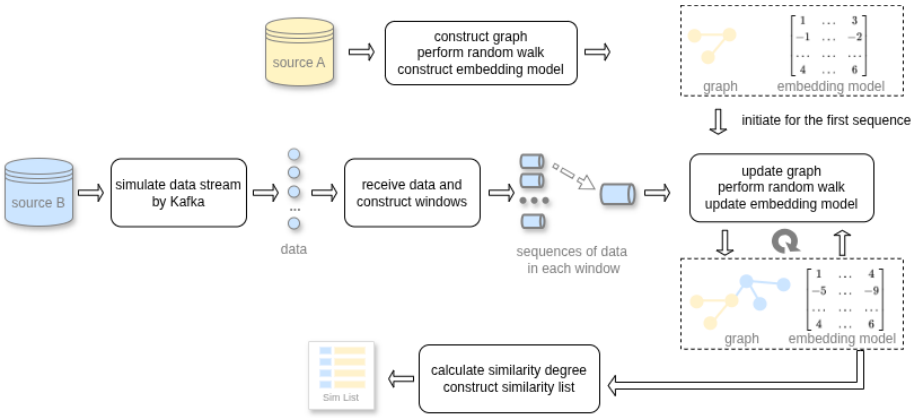


Fig. 2: Pipeline of implementation process

For the machine learning model, we use the Gensim [29] python library to execute word embedding tasks and support online learning. By default, we set configuration parameters as follows: for each token, a vector size of 300 and a context window of 3. During the random walk, we set the walk length to 60 and apply log smoothing to enhance the likelihood of selecting low-probability nodes, thereby expanding the coverage of nodes in the graph by the generated sentences. Considering online learning, we choose the skip-gram algorithm and word2vec model as the main training method. We calculate the inner product of the vectors and obtain the cosine similarity by normalizing the vectors, i.e., dividing their inner product by the product of their norms. Each similarity value is retained to 16 decimal places. Since each random walk introduces variability and generates different sentences for word embeddings, the results of each experiment were averaged across three independent runs to mitigate the influence of randomness on the outcomes.

Experiments have been conducted on a laptop with CPU 12×12th Gen Intel(R) Core(TM) i5-1245U with 15.3 GB RAM.

4.2 Results analysis

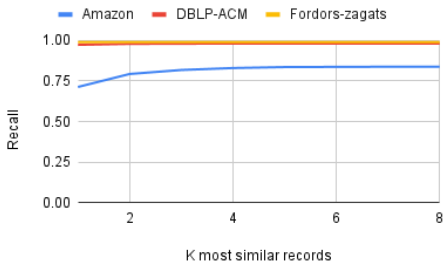
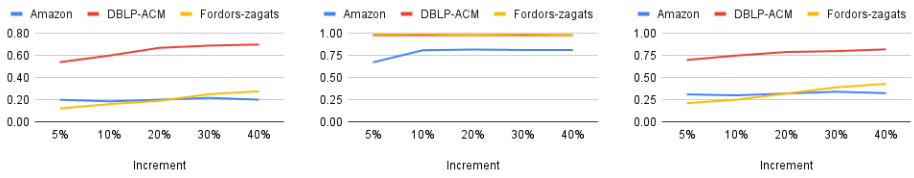


Fig. 3: The effect of the k most similar predicted pairs on the recall rate

Experiments on effectiveness Fig. 3 illustrates the effect of the most similar value ordering on recall, in order to demonstrate the effectiveness of our algorithm. The higher the recall, the more pairs of records are correctly identified. As the value of k increases, we can see that the incremental algorithm successfully predicts most of the similar record pairs. Moreover, most of the correct predictions are concentrated in the records with relatively high similarity ranks.

Adjusting the k-value can effectively enhance the recall for entity resolution tasks. Furthermore, beyond a certain threshold, the length of the similarity list has minimal impact on the accuracy of predicted pairs. Most of the less similar pairs tend to be irrelevant, and focusing on the top candidates with high similarity can significantly improve both the efficiency and accuracy of the algorithm.

Effect of increment size on results. In incremental learning, the size of the newly added data at each step impacts the model's training process. Frequent updates or introducing too much data at once can potentially destabilize the model. To analyze the effect of the incremental data size on the entity resolution task performance, we varied the proportion of the incremental data. The size of the incremental data is described as a percentage of the pre-trained data. For example, a 5% increment means that the amount of data trained in each incremental process is equivalent to 5% of the original pre-trained data volume.



(a) Precision (b) Recall (c) F1-score

Fig. 4: Effect of increment size on results

Fig. 4 illustrates the changes in precision, recall and f1 score as increment size increases. As shown in the figure, the recall improves as increment size increases. This suggests that for the ER task, updating the text corpus too frequently has a greater impact on the results and leads to a decrease in the model's ability to predict similar records. Moreover, different increment sizes influence the metrics in varying ways. These performance indicators tend to stabilize once the increment size surpasses a certain threshold.

The similarity list approach used here does not deeply evaluate dissimilarity, focusing mainly on whether two records are similar. As a result, accuracy may be suboptimal for certain datasets. However, a high recall indicates that the majority of correct pairings are identified, which is crucial in a context where large volumes of unbounded data are rapidly incoming. Reducing the range of potential pairs lays the foundation for subsequent stages that progressively enhance precision, for example, the transition from a streaming processing approach to a more efficient batch processing method for the next steps.

Additionally, the performance differences across datasets, as illustrated in the figures, highlight the significant impact of dataset structure on outcomes. In particular, variations in duplication rates (as shown in Table 2) and the presence of different types of data noise contribute to the variability in results. Noise types such as transposed characters, missing characters, or inconsistent abbreviations reduce textual similarity and increase the complexity of accurate record matching.

5 Conclusion

With the increasing demand for various streaming data analysis, it is essential to integrate streaming data with stored data efficiently. In this study, we addressed the challenge of entity resolution in data integration by adapting existing embedding techniques, traditionally used in batch processing, to a streaming context. We proposed a dynamic solution based on an incremental embedding model that continuously updates representations with the incoming stream of new data.

We construct a heterogeneous graph using relational data and incrementally update it using streaming data sequences. Using this evolving graph, we perform random walks on newly added nodes to generate sentences, which are then used to update the embedding model. Results vary widely across datasets, while our method achieves up to 91% precision, 97% recall, and 94% F1-score, demonstrating its effectiveness in identifying similar matches in streaming contexts.

Nevertheless, our approach focuses primarily on detecting potentially similar record pairs. As a next step, we aim to refine our results using similarity lists to identify mismatches in the list to determine more definitive matches. In addition, since the results vary across datasets, enhancing the adaptability of our method to different data characteristics is essential. We plan to incorporate more specific noise-handling strategies to improve robustness and generalization across diverse scenarios.

References

1. Almeida, F., Xexéo, G.: Word embeddings: A survey (2023)
2. Altwaijry, H., Mehrotra, S., Kalashnikov, D.V.: Query: a framework for integrating entity resolution with query processing. Proceedings of the VLDB Endowment **9**(3), 120–131 (Nov 2015)
3. Bahri, M., Bifet, A., Gama, J., Gomes, H.M., Maniu, S.: Data stream analysis: Foundations, major tasks and tools. WIREs Data Mining and Knowledge Discovery **11**(3), e1405 (Mar 2021). https://doi.org/10.1002/widm.1405
4. Barros, C.D.T., Mendonça, M.R.F., Vieira, A.B., Ziviani, A.: A survey on embedding dynamic graphs **55**(1) (Nov 2021)
5. Binette, O., Steorts, R.C.: (almost) all of entity resolution. Science Advances **8**(12), eabi8021 (Mar 2022)
6. Bravo-Marquez, F., Khanchandani, A., Pfahringer, B.: Incremental word vectors for time-evolving sentiment lexicon induction. Cognitive Computation **14**, 1–17 (01 2022). https://doi.org/10.1007/s12559-021-09831-y
7. Brunner, U., Stockinger, K.: Entity matching with transformer architectures - a step forward in data integration. In: Proceedings of the 23rd International Conference on Extending Database Technology (EDBT) (2020)
8. Cappuzzo, R., Papotti, P., Thirumuruganathan, S.: Creating embeddings of heterogeneous relational datasets for data integration tasks. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. p. 1335–1349. ACM, Portland OR USA (Jun 2020)
9. Chen, R., Shen, Y., Zhang, D.: Gnem: A generic one-to-set neural entity matching framework. In: Proceedings of the Web Conference 2021. p. 1686–1694. ACM, Ljubljana Slovenia (Apr 2021). https://doi.org/10.1145/3442381.3450119
10. Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., Stefanidis, K.: An overview of end-to-end entity resolution for big data. ACM Comput. Surv. (2020)
11. Do Nascimento, D.C., Santos Pires, C.E., Gomes Mestre, D.: Heuristic-based approaches for speeding up incremental record linkage. Journal of Systems and Software **137**, 335–354 (Mar 2018)
12. Ebraheem, M., Thirumuruganathan, S., Joty, S., Ouzzani, M., Tang, N.: Distributed representations of tuples for entity resolution. Proceedings of the VLDB Endowment **11**(11), 1454–1467 (Jul 2018)
13. Efthymiou, V., Papadakis, G., Stefanidis, K., Christophides, V.: Minoaner: Schema-agnostic, non-iterative, massively parallel resolution of web entities. Proceedings of the 22nd International Conference on Extending Database Technology (EDBT) (2019)
14. Gazzarri, L., Herschel, M.: End-to-end task based parallelization for entity resolution on dynamic data. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). p. 1248–1259. IEEE, Chania, Greece (Apr 2021)
15. Gazzarri, L., Herschel, M.: Progressive entity resolution over incremental data. In: Proceedings of the 26th International Conference on Extending Database Technology (EDBT). OpenProceedings.org (2023)
16. Genossar, B., Shraga, R., Gal, A.: Flexer: Flexible entity resolution for multiple intents. Proc. ACM Manag. Data **1**(1) (May 2023)
17. Isah, H., Abughofa, T., Mahfuz, S., Ajerla, D., Zulkernine, F., Khan, S.: A survey of distributed data stream processing frameworks. IEEE Access **7**, 154300–154316 (2019)

18. Kaji, N., Kobayashi, H.: Incremental skip-gram model with negative sampling. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 363–371. Association for Computational Linguistics, Copenhagen, Denmark (Sep 2017)
19. Li, Y., Li, J., Suhara, Y., Doan, A., Tan, W.C.: Deep entity matching with pre-trained language models. Proceedings of the VLDB Endowment **14**(1), 50–60 (Sep 2020)
20. Maharana, K., Mondal, S., Nemade, B.: A review: Data pre-processing and data augmentation techniques. Global Transitions Proceedings **3**(1), 91–99 (2022), https://www.sciencedirect.com/science/article/pii/S2666285X22000565, international Conference on Intelligent Engineering Approach(ICIEA-2022)
21. Mahdavi, S., Khoshraftar, S., An, A.: dynnode2vec: Scalable dynamic network embedding (arXiv:1812.02356) (Feb 2019)
22. Marz, N., Warren, J.: Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co., USA, 1st edn. (2015)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. p. 3111–3119. NIPS'13, Curran Associates Inc., Red Hook, NY, USA (2013)
24. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming **2**(2), 143–152 (1982)
25. Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V.: Deep learning for entity matching: A design space exploration. In: Proceedings of the 2018 International Conference on Management of Data. p. 19–34. ACM, Houston TX USA (May 2018)
26. Papadakis, G., Efthymiou, V., Thanos, E., Hassanzadeh, O., Christen, P.: An analysis of one-to-one matching algorithms for entity resolution. The VLDB Journal **32**(6), 1369–1400 (Nov 2023)
27. Peng, H., Mengjiao, B., Li, J., Bhuiyan, M.Z.A., Liu, Y., He, Y., Yang, E.: Incremental term representation learning for social network analysis. Future Generation Computer Systems **86** (05 2017)
28. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (2014)
29. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. pp. 45–50. ELRA, Valletta, Malta (May 2010)
30. Ren, W., Lian, X., Ghazinour, K.: Online topic-aware entity resolution over incomplete data streams. In: Proceedings of the 2021 International Conference on Management of Data. p. 1478–1490. SIGMOD '21, Association for Computing Machinery, New York, NY, USA (2021)
31. Saeedi, A., Peukert, E., Rahm, E.: Incremental Multi-source Entity Resolution for Knowledge Graph Completion, Lecture Notes in Computer Science, vol. 12123, p. 393–408. Springer International Publishing, Cham (2020)
32. Simonini, G., Zecchini, L., Bergamaschi, S., Naumann, F.: Entity resolution on-demand. Proceedings of the VLDB Endowment **15**(7), 1506–1518 (Mar 2022)
33. Tu, J., Fan, J., Tang, N., Wang, P., Li, G., Du, X., Jia, X., Gao, S.: Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. Proceedings of the ACM on Management of Data **1**(1), 1–26 (May 2023)
34. Wang, S., Zhou, W., Jiang, C.: A survey of word embeddings based on deep learning. Computing **102**(3), 717–740 (Mar 2020)

35. Zeng, X., Wang, P., Mao, Y., Chen, L., Liu, X., Gao, Y.: Multiem: Efficient and effective unsupervised multi-table entity matching. In: 2024 IEEE 40th International Conference on Data Engineering (ICDE). pp. 3421–3434 (2024)