Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Deep Learning in Stream Entity Resolution

Suhas Devendrakeerti Sangolli

Course of Study:	Computer Science
Examiner:	Prof. Dr. rer. nat. Melanie Herschel
Supervisor:	Leonardo Gazzarri, M.Sc., Prof. Dr. rer. nat. Melanie Herschel
Commenced:	January 12, 2022
Completed:	July 12, 2022

Abstract

Entity Resolution (ER) determines which virtual representations of entities map to the same real-world entity. Most current ER-related research in big-data scenarios focuses on *volume* and *variety* problems. However, with increased digitization, data is not only generated in bulk but also in a continuous fashion. So, *velocity* is also an issue that needs to be addressed in the ER domain. Another major issue in the deep learning-based ER is data labelling. It is hard to find pre-labelled data to train the model, and it turns out even more difficult when new data is being streamed continuously.

In this thesis, we aim to address all the aforementioned issues by developing a deep learning-based classification function that incorporates continuous streaming entity pairs and classifies them into match or not-match. The end-to-end system has two main layers; one for training and another for prediction. In the training layer, we use a pre-trained language model (DistilBERT) as a base and train it iteratively as newer entity pairs arrive. To train the model, labelled data are obtained through active learning. The prediction layer makes use of the latest trained model to classify the streaming entity pairs into match or non-match. Both training and prediction layers function in parallel and independent of each other.

We evaluate the system proposed in this thesis on several benchmark datasets that vary in size, skewness and origin-domain. As a evaluation metrics we use F1 score, losses, time and iterations. Our iterative model performs similar to the non-iterative models by achieving a match class's f1 score of 0.97 for benchmark datasets.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

AL Active Learning. 17

BERT Bidirectional Encoder Representations from Transformer. 16

BN Bayesian Network. 24

BSON Binary JSON. 48

CNN Convolutional Neural Network. 26

CP Candidate Pool. 17

CV Computer Vision. 16

DL Deep Learning. 16

EM Entity Matching. 15

ER Entity Resolution. 15

FFNN Feed Forward Neural Network. 46

GPT-2 Generative Pre-trained Transformer 2. 29

HIT Human Intelligence Task. 31

I/O Input-Output. 47

IDF Inverse Document Frequency. 45

IL Increment Learning. 17

JSON JavaScript Object Notation. 15

LP Label Pool. 17

LSTM Long Short-Term Memory. 28

MQL MongoDB Query Language. 48

NER Named Entity Recognition. 30

NLP Natural Language Processing. 16

PLM Pre-trained Language Model. 16

RNN Recurrent Neural Network. 27

SOTA State-Of-The-Art. 23

SQL Structured Query Language. 48

TCP Transmission Control Protocol. 41

TF Term Frequency. 45

TF-IDF Term Frequency with Inverse Document Frequency. 30

1 Introduction

In this chapter, we describe the motivation for our study by providing a brief introduction to the topic of entity resolution and the emergence of deep learning in this domain, which motivated our research. A set of research objectives for this thesis are also defined with a brief description.

1.1 Motivation

"Data is the new oil", says Clive Humby[1]. In the big-data era, many corporations, governments, and scientific institutions rely on vast amounts of data generated from internal and external data sources. Making choices based on these data requires a centralised data warehouse where all internal and external data sources are merged and stored. There are various quality issues with the data integration, such as incompleteness (i.e., partial data), redundancy (i.e., overlapping data), inconsistency (i.e., conflicting data), or simple incorrectness (i.e., data errors). Entity Resolution (Entity Resolution (ER)) is a common activity in addressing these quality concerns [CEP+20].

The problem of automatically determining which virtual representations of an entity belongs to the same real-world entity is ER. Because of multiple schema abstractions, an entity may have distinct virtual representations across different information systems or even within the same information system due to data errors. The goal is to determine which entity pairings in virtual representations correspond to real-world entities. The virtual representation and the real-world entities of the movie domain are shown in Figure 1.1. The triplet, JavaScript Object Notation (JSON), and tabular data formats represent the virtual entities. The entities are compared between the different data formats, thus e_1 is compared to e_4 and e_5. Other entities are compared in a similar fashion. From the entity descriptions, the e_1 and e_5 represents the real-world movie “Sherlock Holmes”. Similarly, e_3 and e_4 represents the movie “Quick Change”.

Entity Resolution (ER) is often divided into three steps: data pre-processing, entity blocking, and entity matching [GH21]. The general ER framework is presented in Figure 1.2. Data pre-processing converts raw input from numerous data sources into a standard format to facilitate subsequent downstream processing. We progress from highly heterogeneous entity representations that use many data formats to more homogeneous entity representations. Entity blocking groups entities based on certain specifications (blocking-key). This stage primarily seeks to reduce the total number of comparisons required in the Entity Matching Entity Matching (EM). Entity pairs are created from the same cluster and fed into EM. Different comparison strategies (Non-learning-based, Learning-based, and others) are used in EM to compare entity pairs and label them as match or not-match. Our thesis tries to solve the EM problem using learning-based strategies.

[1]Clive Humby: https://en.wikipedia.org/wiki/Clive_Humby

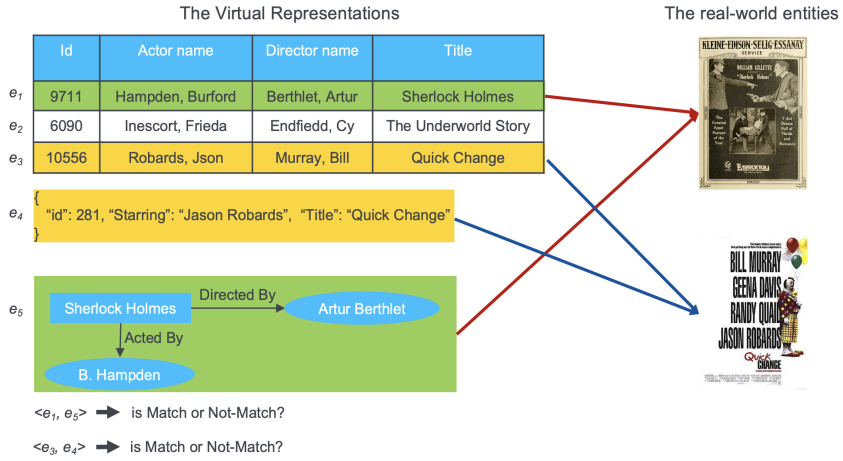


Figure 1.1: The virtual representation of real-world entities.

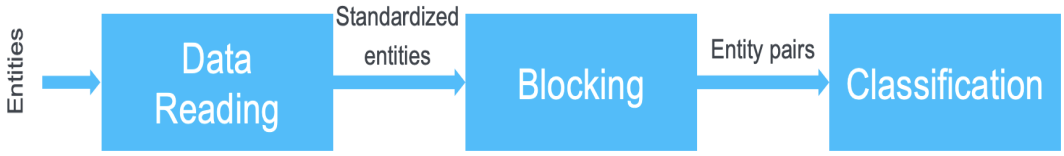


Figure 1.2: The general framework of ER [GH21].

According to [DS15][CEP+19], modern ER should deal with the four "V's of big data"(Volume, Variety, Velocity, and Veracity). Because entity pairs are delivered as streams in the context of big data, incremental techniques for ER can be used to address the Volume and Velocity issues. Blocking approaches for efficient incremental ER are still in their infancy [CEP+19]. Recent work [GH21] demonstrates that it is possible to execute incremental blocking on extremely unstructured data such as DBpedia, with millions of entities and hundreds of attributes, while meeting low latency and high throughput requirements. But there is a requirement for incremental ER based classification functions that can classify the continuous streaming entity pairs.

Traditional ER approaches identify records via pairwise comparisons, which presume the existence of a distance function that provides information about the similarity of entity pairs [GM12]. In dealing with traditional entity resolution throughout the last few decades, rule-based techniques [LLG14] and hand-crafted heuristics have proven critical. Traditional ER approaches work well with structured data, but they perform poorly with unstructured and semi-structured data [Moh20]. Learning-based (Machine learning, Deep learning) ER has gained popularity in recent years to address these difficulties. Some machine learning-based ER algorithms have shown significant gains in accuracy [LLS+20][MLR+18][MS04][CKLS01].

Deep Learning (DL) has lately been recognized as a crucial component of several domains concerned with unstructured data, including Computer Vision (CV) and Natural Language Processing (NLP). Its main benefit over previous approaches is its capacity to learn features rather than depending on manually built features. Pre-trained Language Models (PLMs) have received a lot of interest in text processing and NLP in recent years. PLMs are huge neural networks that function on a pretrain-finetune paradigm, which means that models are first pre-trained on a large text corpus and then fine-tuned to solve specific downstream tasks. One such cutting-edge PLM is Bidirectional

Encoder Representations from Transformer (BERT) [DCLT18]. It has been utilized in a variety of NLP applications, including question answering [RZLL16] and sentiment analysis [SPW+13]. To fine-tune such PLMs, however, a huge amount of labelled data is required. Even fine-tuned PLMs become obsolete over time as the complexity and variety of input data supplied into such models increases. Active Learning (AL) and Increment Learning (IL) are being researched to address these concerns.

In this thesis, we demonstrate the design and implementation of a DistilBERT-based entity matching component that can be integrated into an incremental ER system and function effectively on streaming data. This solution includes a matcher as well as two pools: the Candidate Pool (CP) and the Label Pool (LP). The input pairs from the entity blocking are classified using matcher (DistilBERT) and stored in the CP. AL and crowd-sourcing-based ER methods are used in this thesis to increase model performance. The human oracle retrieves entity pairs from the CP, labels them as matches or not-matches, and updates the LP using an acquisition function. Following the update, the system fine-tunes the DistilBERT [SDCW19] model asynchronously using the LP, and the trained model is updated in the matcher. In this manner, the model is incrementally learned.

These aspects mentioned above of the thesis makes certain research contribution to different topics. They are segregated into unique research objectives and described in detail in the following sections.

1.2 Research Objectives

1. **To address the problem of entity resolution for streaming entity pairs.**
   Most of the existing DL-based ER work incorporates only batch data for ER problems. As part of this research objective, we are going to implement a classification function that classifies the continuously streaming entity pairs into a match or not-match class. To classify every input entity pair, the latest trained classification function is used.

2. **To explore the concept of continuous iterative training to achieve ER in the case of continuous streaming entity pairs.**
   In order to achieve the classification of continuous streaming entity pairs with the latest classification function, it needs to be trained with the newer dataset, with continuous iterative learning. So, as a part of this research objective, we investigate different methods that can be applied the same way or that can be extended to accomplish continuous iterative learning. A classification function that does not require the entire training data set or data entries of the same schema is designed and implemented.

3. **To explore the role of active learning and crowd-sourcing-based ER in incremental model training.**
   Active Learning and Crowd-sourcing-based ER are key aspects of incremental model learning as they provide true labels of entity pairs for model learning. In this thesis, we explore different options and implement a suitable method to provide these true labels for incremental model training.

4. **To evaluate the end-to-end streaming entity resolution system developed in this thesis and analyze its performance.**
   To assess the performance of the system, it is subjected to different conditions and its

performance is measured based on certain key aspects such as F1 score, Training and Testing losses, and the number of iterations required for convergence. These conditions involve variation in the input dataset with respect to sample size, skewness, and domain. Multiple experiments are run for every condition with the end-to-end system and findings are visualized to derive meaningful insights.

1.3 Thesis Structure

The rest of this thesis is organized into multiple chapters and their contents are as follows.

Chapter 2 discusses the preliminaries and related work, which covers the existing research on topics related to or being directly addressed in this thesis. **Chapter 3** presents an architectural overview of the system and provides brief explanation on how the entire system works, starting from receiving the input entity tuple stream to producing classified output. **Chapter 4** explains the concepts behind implementation of training and prediction layers and describe their implementation in detail. Challenges faced during the implementation of both the layers and their limitations are also presented. As part of the evaluation of the developed ER system, **Chapter 5** presents design and findings of the experiments conducted. Results of the experiments are visualized and inferences of those results are presented. Finally, **Chapter 6** concludes this thesis by revisiting the research objectives and future work is discussed.

2 Preliminaries and Related Work

Entity Resolution (ER) is the challenge of translating several entities into a single real-world entity. This section gives a comprehensive description of entity resolution responsibilities and potential obstacles. Before proceeding, it is essential to know that the literature also refers to ER as Record Linkage [FS69], and Entity Matching [KR10] among other terms.

2.1 Entity Resolution

An entity exists as itself, a subject or object, physically or hypothetically[1]. Entities used to refer to individuals, such as patients, customers, or taxpayers, but they may now also refer to publications, citations, customer items, or enterprises. An entity may have distinct virtual representations across different information systems due to differing schema abstractions or even within the same information system as a result of data faults. ER, as defined in [CEP+19][Chr][Tal11], is the challenge of determining which virtual representation of an entity corresponds to the same real-world entity. In other words, ER is also described as the job of recognising, matching, and merging entries from several databases that relate to the same entities [ETJ+18]: Consider T to be a collection of entities with n tuples and m attributes $A_1, ..., A_m$. It should be noted that these entities might come from a single database or numerous tables (with aligned attributes). The value of attribute A_i on tuple t is denoted by $t[A_i]$. Given all distinct tuple pairs (t, t') from T where $t \neq t'$, the problem of ER is to determine which pairs of entity tuples refer to the same real-world entities. These attributes serve as the entity's features. These features are utilised to address ER problems [Moh20]. Figure 2.1 is an example of entity resolution as given by [CES15]. This Figure shows movie information from DBpedia[2] and Freebase[3] data sources. The entity structure is sufficient to categorise the movie "Eyes Wide Shut" from DBpedia and Freebase databases. However, in the instance of the director entity, the entity structure is difficult to categorise; a more relevant entity structure, such as birth location or birth date, is necessary.

ER is an essential part of many real-world applications. The health department is the earliest area that has established ER procedures to handle patient data duplication [Chr]. National censuses collect statistics on population, culture, economics, and environment in their respective countries. The cost of merging various data sources is reduced using ER approaches [Gil01]. Another use of ER is in detecting crime and fraud, where it is critical for recognising criminal's altered or fictional personal information [BBS05]. Recently, ER approaches have been utilised to compare items from different sellers in online shopping and E-Commerce [BBS05]. Many firms are shifting away from

[1] Entity: https://en.wikipedia.org/wiki/Entity
[2] JedAI dataset: https://github.com/scify/JedAIToolkit/tree/master/data
[3] Freebase Triples: https://developers.google.com/freebase

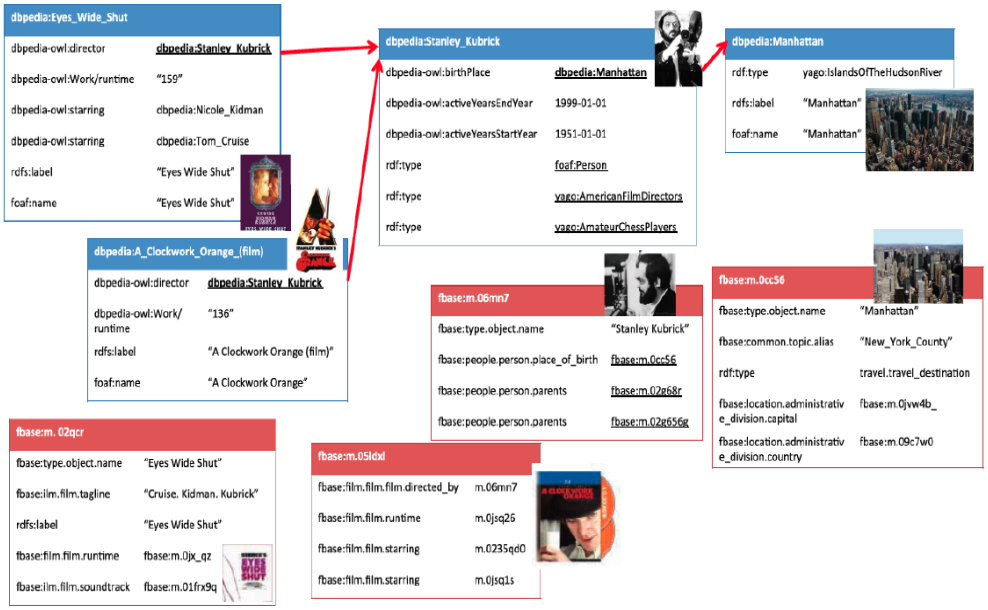


Figure 2.1: The movies ER example [CES15].

multiple data sources to centralised data storage. ER is used to address data integration issues such as incompleteness (i.e., partial data), redundancy (i.e., overlapping data), inconsistency (i.e., conflicting data), and simple incorrectness (i.e., data mistakes) [CEP+20].

2.2 Entity Resolution Generations

As data volumes have increased in recent years, ER has emerged as the primary study field in computer science, with several ER algorithms established thus far. **Papadakis et al.** [PIP20] examine the evolution of ER approaches and the associated problems. These ER generations are discussed below.

1. **Veracity**: Inconsistencies and inaccuracies in entities were a challenge that ER methods were first developed to address. These issues are brought on by machine or human data-entry limitations [EIV06]. The Figure 2.2 shows the first generation of ER workflow. The entity descriptions for schema matching are mapped using a mapping function based on the structure, name, and values [BMR11] [MBR01]. A quadratic time complexity results from the pair-wise comparison of entities. The blocking technique, which groups entities based on blocking keys such as hash-based keys [FS69], learning and non-learning-based functions, is developed to address this issue. By pairing entities from the same group, blocking reduces the complexity [Chr12]. For example, entity pairs are grouped in the movie domain if the same director directs both movies. The entity pairs are provided as input to the entity matching step. Learning-based or non-learning-based approaches [KTR10] are then employed to find the similarity score between the entity pairs. Based on this similarity score, the entity pairs are classified into three categories: match, non-match, or unsure [Chr12].

Figure 2.2: First and Second generation ER workflow [PIP20].

2. **Volume**: The quantity of the input data, which comprises millions and billions of entities, is the next major hurdle. To overcome this issue, the same first-generation ER methodology is used. Several approaches, including [KTR12][BDNW12], make extensive use of enormous parallelisation, such as Map/Reduce [DG08] in blocking and EM.

3. **Variety**: The third generation ER approaches addresses the issue of variety along with veracity and volume, i.e., heterogeneity or errors in vast amounts of data. Variety was produced by a lack of centralised data management, schema, semantics, and the problem of data noise [CES15][DS15]. Figure 2.3 depicts the third-generation ER process. Schema clustering aims to partition the large clusters into smaller ones by adopting the attribute values and blocking keys. This method is used as blocking in attribute clustering [PIP+12] and meta-blocking with BLAST [SBJ16]. Blocking step is divided into two sub-steps: block building and block processing. Block building adopts the attribute value and discards the attribute name to construct blocks. Token blocking [PN11] is the essential block-building method, which creates the blocks for each token of the attribute value of the entity. This creates the problem of multiple blocks and, as a result, increases the number of comparisons. To resolve this issue, the block processing step is adopted. This step consists of calculating the match likelihood score for each entity pair and deleting the entity pair with the lowest score. The entity matching step then compares each entity pairs from the final blocks. The entity matching step is an iterative step in which the match likelihood score is calculated for initial match entity pairs and then iterates through the neighbours, by updating their scores. Then the entity pairs are sorted by matching likelihood score, and the pair with the highest score is labelled as the match. This approach is used in most of the significant entity matching algorithms such as SiGMA [LPD+13], PARIS [SAS11], and LINDA [BDNW12].

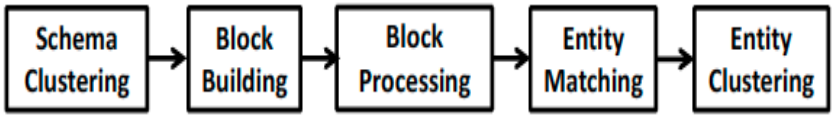


Figure 2.3: Third generation ER workflow [PIP20].

4. **Velocity**: The fourth-generation ER approach addresses velocity, as well as veracity, volume, and variety. This is due to the continuous generation of data, which brings unique ER issues, such as the collected data is not definitive and newly generated data may affect the collected data. To solve these problem, progressive ER generates relevant outcomes in a pay-as-you-go manner prior to the completion of ER. The schema-aware [PHN14] and schema-agnostic [SPPB18] processes are shown in Figures 2.4(a) and 2.4(b). Another approach that minimizes the expense of changing old findings when new evidence becomes available is incremental ER [GDS14]. As a incremental ER solution, Gazzarri et al. [GH21] investigate two approaches:

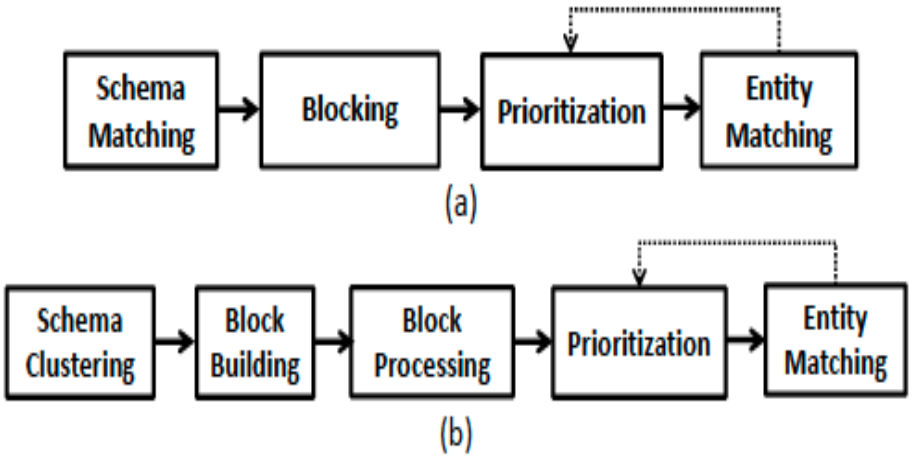


Figure 2.4: The end-to-end workflow of the fourth generation ER for (a) structured and (b) semi-structured data [PIP20].

(1) incremental ER, in which a limited data-set updates on a regular basis, keeping complete ER findings progressively, and (2) streaming ER, in which a potentially endless stream of entity descriptions is analyzed.

2.3 Entity Resolution methods

Entity Matching (EM) function M is a basic ER function that determines each entity pair description as $\{match,\ not-match\}$, with $M(e_a, e_b) = match$ indicating that e_a and e_b map to the same real-world entity, and $M(e_a, e_b) = not-match$ indicating that e_a and e_b map to separate real-world entities. M is described as a similarity function sim, which measures how similar two entity descriptions are based on certain comparison methods. This sim function may be made up of individual functions, such as jaccard similarity[4], or a composite one, such as a linear combination of numerous similarity functions on distinct aspects of descriptions [CEP+20]. A simple similarity function could be, given a threshold θ,

$$M(e_a, e_b) = \begin{cases} true, & \text{if } sim(e_a, e_b) \geq \theta \\ false, & \text{otherwise} \end{cases} \tag{2.1}$$

It is very hard to find a similarity measure that can completely differentiate all matches from not-matches using simple pairwise comparisons on the attribute values of two descriptions [CEP+20]. In practice, most of the similarity functions aim to reduce the number of incorrectly categorized pairs. Different matching functions might be considered depending on the structuredness and type of comparisons, as well as the availability of known, pre-labelled matching pairs. These are primarily divided into two categories: non-learning-based and learning-based methods [CEP+20].

[4]Jaccard Index: https://en.wikipedia.org/wiki/Jaccard_index

We concentrate more on learning-based existing research in the following sections since we employ learning-based ER algorithm. We also go through a non-learning based State-Of-The-Art (SOTA) method.

2.3.1 Non-learning-based ER Method

MinoanER [EPSC19] presents a non-learning-based matching process that is implemented in Spark [ZCF+10], with the matching process using a certain number of predefined generic, schema-agnostic matching rules that traverse the blocking graph. This approach takes the disjunctive blocking graph as input and applies four matching rules: the *Name Matching Rule (R1)*, the *Value Matching Rule (R2)*, the *Rank Aggregation Matching Rule (R3)*, and the *Reciprocity Matching Rule (R4)*. Figure 2.5 depicts the MinoanER architecture in Spark.

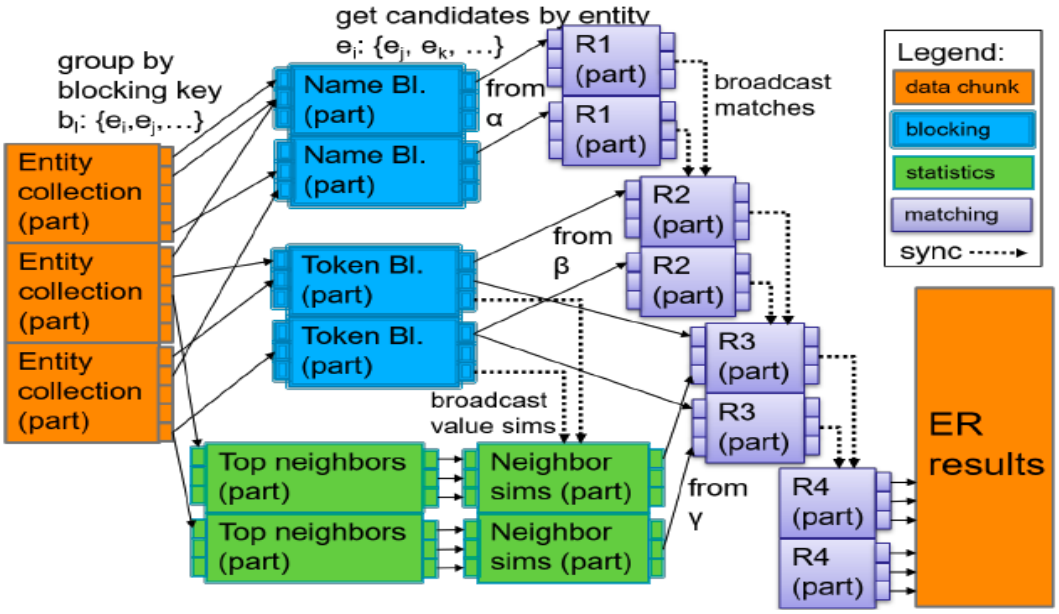


Figure 2.5: The architecture of MinoanER in Spark [EPSC19].

R1 proposes that two candidate entities are matched if and only if their names are identical. The remaining rules do not apply to any of the candidates that match R1. The *R2* assumes that the two entities match if and only if they have a common token or a large number of rare tokens. It identifies descriptions that have high-value commonalities. R2-identified matches will not be considered in the sequel. Where the order of candidates is employed rather than the absolute similarity value, *R3* indicates potential matches for candidates whose value similarity is low but their neighbour similarity is reasonably high. The *R4* tries to clean the matches identified by R1, R2, and R3 and enhance the accuracy based on the logic that two entities are unlikely to match when one does not even consider the other to be a possibility for matching. Only if both entity descriptions accept that they are likely to match.

2.3.2 Learning-based ER Methods

In the first probabilistic ER model [FS69], the properties of entity representations were employed as parameters of comparison vectors. Following this, several forms of study were conducted in order to automate the ER.

Gradient-based matching [RPHP17] presents a supervised gradient-depending learning model that can alter its structure and parameters based on matching scores from various comparison functions. It may be used in a wide range of domains to successfully classify data. Figure 2.6 depicts the appropriate ER procedure.

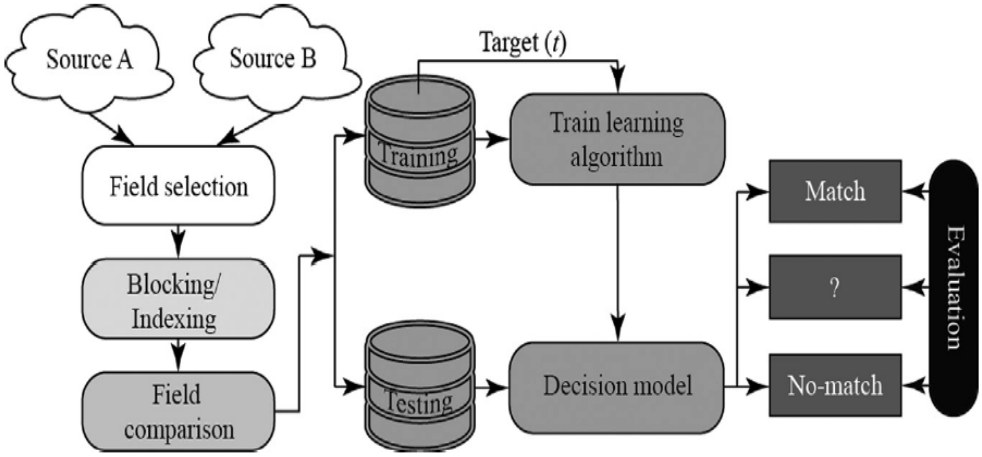


Figure 2.6: Architecture of the Gradient-Based Matching.

The approach considers entities from various sources as inputs, and then fields such as name, address, and date of birth (DOB) are examined for indexing. The indexing step lowers the data matching process's quadratic complexity. The candidate record pairs from the indexed data structures are compared using specified field comparison functions in the second stage. The second phase divides the comparison scores into training and testing sets. The model's performance is evaluated in the last step.

The gradient-based matching technique uses Soundex [OR18] to sort keys that will aid in grouping comparable records during the indexing process. It presents two model learning approaches: (1) using various comparison functions to the same field and (2) applying the same comparison functions to other fields. This method helps study the structure and determine which fields and comparison functions to aid in overall categorisation. This system uses the gradient descent back-propogation algorithm [Žil06] as an optimization technique, combined with sigmoidal activation functions for the nodes.

The authors of **BN-Based ER** [INN08] propose a Bayesian Network (BN) based ER that computes the probability obtained in the information space. The important aspects for determining matches are the similarity of text values and the connection between participating entities. The problem formulation incorporates an information space consisting of metadata of entity description. The metadata and accompanying BN are shown in Figure 2.7. The supporting evidence (probability) for each item in the information space is calculated using BN in this technique.

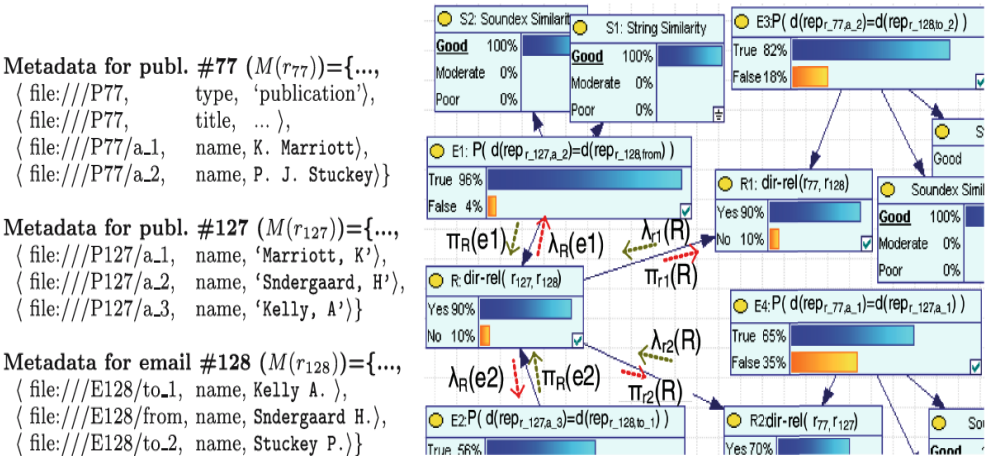


Figure 2.7: Left: Metadata for desktop resources and Right: Corresponding bayesian network [INN08].

A BN [JN07] is a probabilistic graphical model that describes a set of variables and their conditional interactions using a directed acyclic graph. The graphs constructed using BN-based ER have the following nodes. *Entity Nodes* reflect a match in the network using probabilistic inference based on the cause-effect connection. By comparing the literals of the entities, the *Evidence Node* represents evidence for entity nodes. *Direct-Relation Nodes* are the result of entity nodes and deductive-relation nodes, and they depend on the idea that two resources are associated when their descriptions include the same item. *Deductive-Relation Nodes* reflect an indirect relationship between two resources that may be deduced by merging the information of two nodes, either direct-relation or deductive-relation. These nodes form the BN, which is also utilized to adjust to incremental ER.

The DL-based ER methods are discussed in the next section.

2.4 Deep Learning based ER Methods

Deep learning has recently been embraced by several ER systems owing to its capacity to learn features rather than depending on manually built features [LBH+15].

DEEP SEQ-to-SEQ ER [NHH+19] introduces a novel entity resolution approach, in which an align-compare-aggregate neural network is presented, which can learn the representation of a token and collect matching evidence for precise end-to-end ER decisions. Each entity pair is thought of as a sequence of tokens, with each token consisting of an attribute and a word as <attribute, word>.

Figure 2.8 depicts the detailed structure, which is made up of numerous layers. Each token is embedded in low-dimensional vectors in the representation layer so that it may be compared and aligned in the subsequent levels. Concatenating the word and attribute embeddings yields token embedding. FastText [BGJM17] is chosen in this work because it handles the out of vocabulary terms via character-level representation. The alignment layer is used to determine token correspondence. To calculate the relationship between distinct tokens, an attention method is utilized. An alignment matrix is built from normalized attention ratings between token representations using neural attention [BCB14]. The comparison layer's goal is to provide a succession of matched signals.

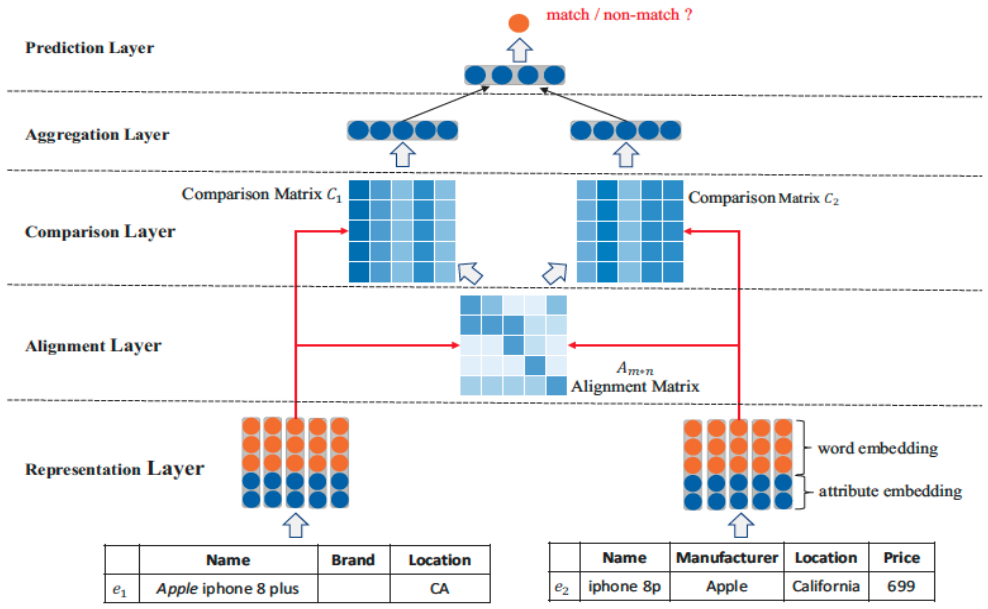


Figure 2.8: Deep Seqence-to-Sequence ER model framework [NHH+19].

A soft-attended representation for each token in one entity is calculated utilizing all tokens in another entity. To achieve this representation, k-max weighted attention is applied, which maintains important pieces while rejecting irrelevant pieces. A Convolutional Neural Network (CNN) [Che15] module is employed in the aggregation layer to aggregate the matching signals from two comparison matrices produced by both directions of the comparison phase. Convolution and max-pooling are two successive procedures in this process. Finally, the prediction layer assesses similarity based on the two feature vectors created in the preceding layer. To get the final similarity measure, these two vectors are synthesized and then fed through a two-layer fully connected neural network followed by a softmax classifier. The similarity criterion is set to 0.5; if the prediction exceeds the threshold, entity pairings are labeled as a match; otherwise, they are labeled as not-match. The findings reveal that seq2seq performs well on nine common ER benchmarks. This technique outperforms DeepMatcher [MLR+18] and Magellan [KDD+16] in solving diverse and dirty ER issues by up to 14.5%.

DEEP-MATCHER [MLR+18] is an architectural template for integrating several DL models into an entity matching problem. Figure 2.9 depicts this template, which is separated into three primary modules: attribute embedding, attribute similarity representation, and classification. The attribute embedding module takes a string of words and turns it into two strings of word embedding vectors, the members of which correspond to d-dimensional embeddings of the words. FastText [BGJM17], word2vec [MSC+13], and GloVe [PSM14] are some possibilities for attribute embeddings. The attribute similarity representation module receives these embedded vectors as input. The primary purpose of this module is to learn a representation that captures the similarity of two embeddings automatically. This procedure is divided into two parts: attribute summarization, in which the module applies a summarization function H to summarize the information in the input sequence and attribute comparison, in which the summarized vectors are taken as input and a comparison function D is applied to those summaries to obtain the final similarity representation of the attribute

values. The classifier module is the last module, which takes the similarity representations as input and utilizes them as features for a classifier *M* that assesses if the input entity descriptions are match or not-match.

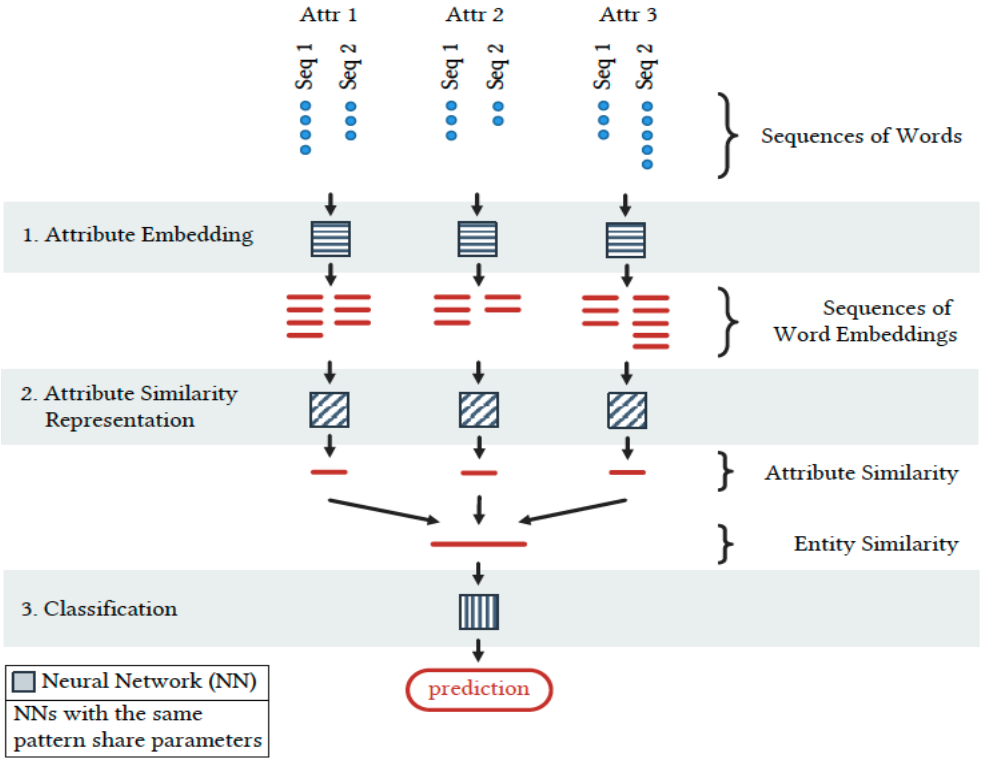


Figure 2.9: Deep Matcher architecture template for DL solution for EM [MLR+18].

This work also presents a classification of DL solutions for a variety of matching problems, as well as a design space for these solutions. These solutions are separated into four sections. The *SIF: An aggregate function model* that forms the input to the classifier module using aggregate functions, especially a weighted average motivated by the Smooth Inverse Frequency (SIF) phrase embedding model [ALM17]. *RNN: A Sequence-aware model* that employs a bidirectional Recurrent Neural Network (RNN) [SP97] for attribute summarization. This model is made up of two RNNs: a forward RNN that processes the input word embedding in normal order, and a backward RNN that does the same but in reverse order. As a result, the final attribute summary representation is the concatenation of the bidirectional RNN's last two outputs. *Attention: a sequence alignment model*, that implements attribute summarization and attributes comparison via decomposable attention [PTDU16]. Hybrid: Sequence-aware with attention model, which forms the input to the classifier module using a bidirectional RNN with decomposable attention for attribute summarization and a vector concatenation with an element-wise absolute difference during attribute comparison.

It also divides EM issues into three types: structured EM, textual EM, and unclean EM. According to the results of the study, DL does not beat existing EM solutions on structured EM, but it may greatly exceed them on textual and filthy EM.

DEEPER [ETJ+18] is a large entity resolution system that extracts tuple embeddings from single word embeddings with high accuracy, efficiency, and usability. It proposes an approach that uses substantially less labeled data by taking prior knowledge of matched values into account, and it can capture both syntactic and semantic similarities without the requirement for feature engineering. The

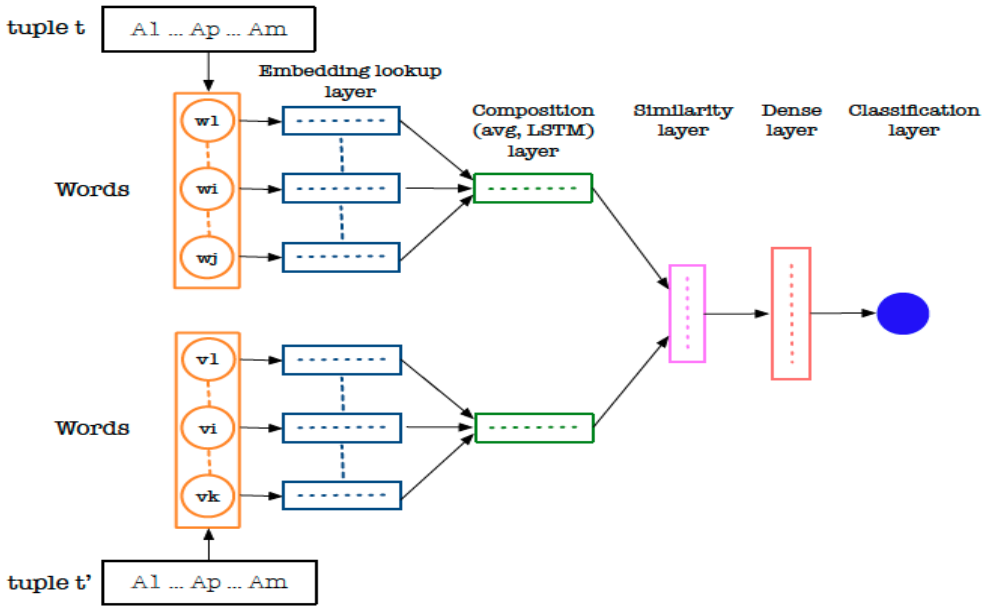


Figure 2.10: DeepER framework [ETJ+18].

DeepER structure is shown in Figure 2.10, which comprises the *Embedding lookup*, *Composition*, *Similarity*, *Dense*, and *Classification layers*. The *embedding layer* is used to embed tokens into d-dimensional vectors. These embeddings are integrated into a single vector in the *Composition layer*. DeepER generates a distributed representation of tuples (DR) using two methods: a *simple average* and a *compositional technique*. In the first approach, the vector representation for an attribute value is generated by averaging the appropriate embedded vectors of value tokens. In the latter, uni- and bi-directional RNNs with Long Short-Term Memory (LSTM) [HS97] hidden units are used. An LSTM cell has specifically designed gates for storing, changing, or deleting information, which enables RNNs to learn a broad range of sequential dependencies and exhibit dynamic temporal behaviour. Bidirectional RNNs [SP97] detect dependencies in both directions, yielding two different interpretations of the same sequence. As a final representation of the attribute value, the two vectors from bidirectional RNN are concatenated. The *dense layer's* similarity is used to compare the DRs of the tuple; comparable words are close to one another in their semantic space. The *classification layer* classifies tuples as matching or not matching.

2.5 Pre-trained Language Model based ER Methods

Pre-trained language models PLMs, according to [EKR+21], are massive neural networks that perform a broad range of NLP tasks. They follow a pretrain-finetune paradigm: Pretraining models on a large text corpus is followed by finetuning on a downstream job.

PLMs, such as BERT [DCLT18] and Generative Pre-trained Transformer 2 (GPT-2) [RWC+19], are DL networks with numerous transformer layers [VSP+17], often 12 or 24, that have been trained in unsupervised fashion on big corpora such as wikipedia articles. The model is self-trained during pre-training to perform auxiliary tasks such as missing tokens and next-sentence prediction. After pre-training, studies [CKLM19][TDP19] reveal that the shallow layers capture lexical meaning while the deeper levels collect syntactic and semantic information.

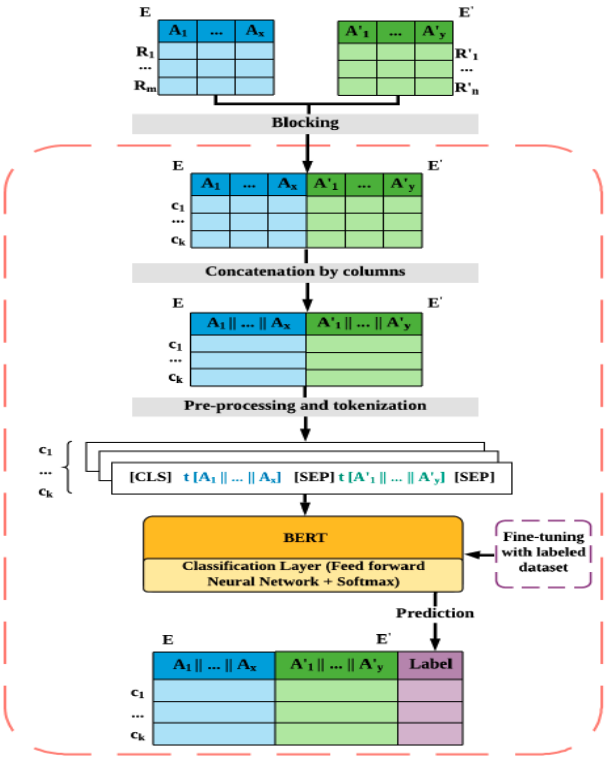


Figure 2.11: Architecture of Schema-Agnostic-ER [TSS20].

Schema-agnostic-ER [TSS20] offer a schema-agnostic entity matching method based on PLMs. The notion of treating tuples in tables for EM, analogous to the sentence pair classification issue in NLP, supports this research. By employing BERT, it mainly tackles the issue of schema-agnostic data, i.e., the input does not consist of the same schema for all data points, and this study only gives a solution in the matching stage of ER.

Schema-agnostic-ER presented a schema-agnostic EM design in Figure 2.11. The candidate pairings from multiple data sources are examined in the first phase of architecture, and the attributes and their related values are concatenated to remove attribute boundaries and construct a phrase. These concatenated entity pairs are then tokenized and pre-processed so that the BERT model can interpret them. In the pre-processing stage, special tokens such as $[CLS]$ are added to specify the classification task to BERT, and $[SEP]$ serves as a separator token to indicate the end of a sentence. These pre-processed tokens of entity pairs are sent into the BERT model, which generates the entity pairs' distributed representation (DR). The DR is fed into the feed-forward neural network and softmax to determine if it is a match or not. According to the testing data, this method beats DeepMatcher and Magellan by an average of 9% in F1 score.

DITTO [LLS+20] is another EM approach that employs PLMs. It presents a revolutionary EM solution based on PLMs like BERT. It alters and recasts EM as a sequence-pair classification issue in order to use such models with a simplified architecture. Figure 2.12 demonstrates DITTO's architecture, which includes multiple serialization and optimization approaches. The serialization stage transforms the input entity descriptions into "$[COL]attr_1[VAL]val_1...[COL]attr_k[VAL]val_k$", where $[COL]$ and $[VAL]$ are special tokens used to indicate the beginning of attribute names and values, respectively. Other serialization strategies include deleting special tokens $[COL]$ and $[VAL]$, as well as omitting attribute names that are unnecessary. This strategy is equally applicable to the heterogeneous schema. The fundamental contribution of the DITTO to EM is three optimization techniques: *Domain Knowledge*, *Summarization*, and *Augmentation*.

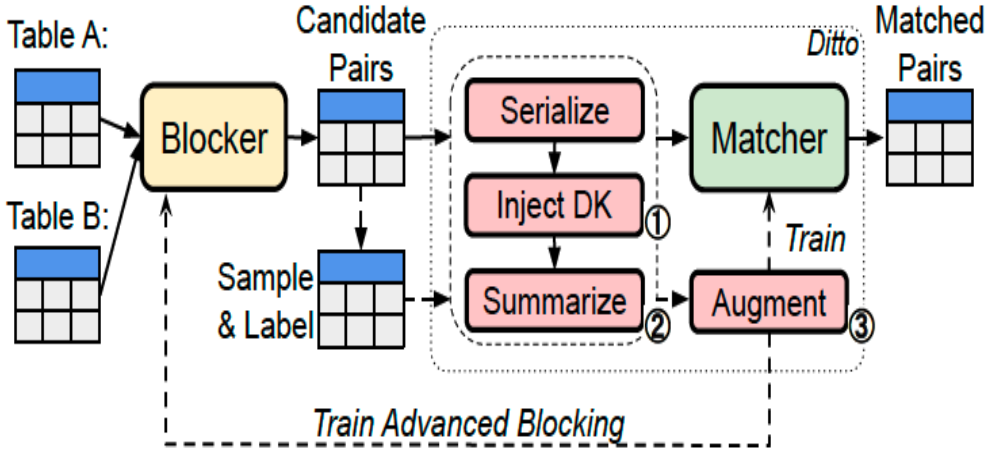


Figure 2.12: Ditto architecture. (1) Domain Knowledge, (2) Sumarization, and (3) Augmentation [LLS+20]

The goal of *Domain Knowledge* (DK) is to highlight what knowledge is possibly essential for EM. DITTO offers two kinds of DK: Span Typing and Span Normalization. The span of the token is regarded as one kind of DK in the former type, where the developer defines a recognizer. The recognizer accepts a text string as input and returns a list of span start/end positions as well as the kind of span. It uses the Named Entity Recognition (NER) paradigm to recognize known kinds such as people, dates, and organizations, and it employs regular expressions to identify particular types such as ProductID and the last four digits of a phone number. In the input, these new tokens are substituted. Span Normalization rewrites syntactically distinct but equivalent spans into the same text in the latter. To do this, the developer provides a set of rewriting rules for spans, such as rounding all floating-point figures to two decimal places and removing all commas for integers. Because the BERT model's input can only include 512 sub-word tokens, the *Summarization approach* is used to summarize extraordinarily large strings. It employs a Term Frequency with Inverse Document Frequency (TF-IDF)-based summarizing approach that maintains non-stop word tokens with high TF-IDF scores. The input sequence is supplemented by numerous operations in the *Augmentation approach*. Some of them are span_del, which will randomly remove the span of tokens of length at most four without special tokens, span_shuffle, which will sample a span of length at most four and randomly shuffle the order of its tokens, and attr_del and attr_shuffle, which are similar operations.

DITTO studies demonstrate that it increases matching quality greatly and surpasses earlier SOTA approaches by up to 29% of the F1 score on the benchmark data-set. The three optimization strategies improve system performance by up to 9.8%.

2.6 Incremental and Crowd-sourcing-based ER Methods

2.6.1 Incremental-based ER Methods

Some big-data operations need the resolution of descriptions that enter in high-velocity streams or as queries against a known entity collection [CEP+20]. Rather than performing a static, offline procedure across all available entity descriptions, such applications handle as many entities as necessary as long as individual descriptions are resolved in near-real-time.

Figure 2.13: Overview of task-based parallelization ER functionalities [GH21].

Task-Based Parallelization ER [GH21] presents task parallelism for ER, which supports incremental ER, which allows continuous calculation of the solution by streaming results of an intermediate step of ER as soon as they are calculated. This work focuses on Big-Data processing, namely data received from diverse Web sources that are very diverse and semi-structured. Figure 2.13 depicts a high-level overview of the framework's features. The pre-processing stage is similar to the general framework's pre-processing step. The pre-processing function depends on each entity's representation individually, i.e., the function may be applied to entities independently of other entities. Token blocking processes each entity description independently to identify which blocks an entity description belongs to. The block cutting function uses incoming block ids and entity pairs to progressively expand blocks and emit pairs of entity representation until the block reaches a predetermined size. Pairwise comparisons are pruned as soon as they are duplicate during the comparison cleaning stage. This is accomplished by determining if an input entity representation pair has already been detected and if so, pruning it from processing to the following stages. In the pairwise comparison stage, a similarity score is produced for each received pair. The classification stage labels the input entity pairs as matching or not matching. If the pairings match, they are sent to the clustering stage, where they are added to duplicate clusters.

2.6.2 Crowd-sourcing-based ER Methods

Crowd-sourcing is a science that investigates methods of delegating difficult work, known as Human Intelligence Tasks (HITs), to humans [CEP+20]. This platform provides a more precise, yet costly and time-consuming method of bringing people into the process. Crowd-sourcing-based ER represents that people may increase matching's efficacy by using contextual knowledge and common sense.

CrowdER [WKFF12] presents a hybrid human-machine technique in which computers do a first, coarse sweep through all data and humans check just the most probable matching pairings. It creates a heuristic-based method for producing cluster-based HITs and analyses it theoretically and experimentally using actual data sets and major crowdsourcing platforms. The hybrid human-machine process is shown in Figure 2.14.

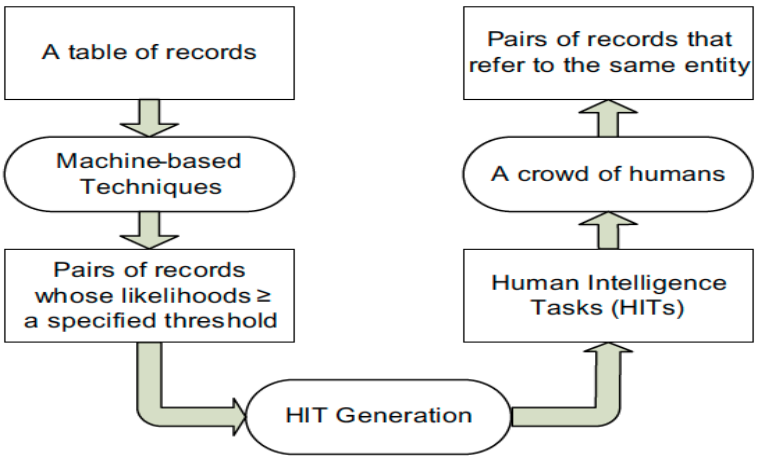
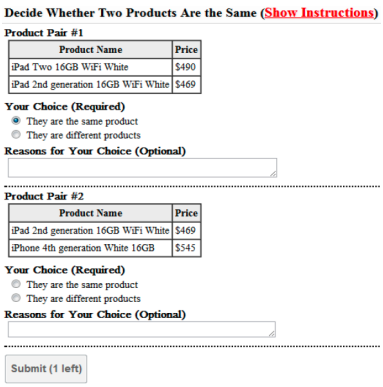
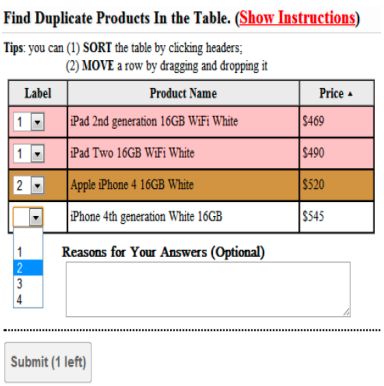


Figure 2.14: Hybrid Human-Machine workflow [WKFF12].

A table of records is initially sent into machine-based ER approaches, which use two strategies: similarity-based and learning-based. In similarity-based approaches, the entity pairings are sent into a similarity function, such as jaccard or cosine similarity, which returns the similarity score. Entity pairs are represented as a feature vectors in learning-based, with each dimension representing a similarity value of the records on some attribute. To train the classifier, learning-based algorithms need a training set. CrowdER recommends two HIT generation techniques: pair-based and cluster-based. The audience verifies several pairs of records as matching or not matching in pair-based HIT creation. Each HIT shown is made up of two pairs of records. The person must choose either "They are the same product" or "They are different products" for each pair. Cluster-based HIT creation is made up of a collection of individual records rather than pairs. The employees are then tasked with locating all of the duplicates in the cluster. The clusters are generated in such a manner that humans



(a) Pair-based HITs



(b) Cluster-based HITs

Figure 2.15: User Interface of CrowdER [WKFF12].

determine the number of clusters and each record belongs to at least one cluster. Goldschmidt et al. [GHHY96] approximation approach is employed to enhance cluster-based HIT creation. Figure 2.15(a) and Figure 2.15(b) depict the user interfaces for pair-based HIT and cluster-based HITs, respectively.

2.7 Summary of Related Work

In this chapter, we discussed the generations of ER by studying the paper [PIP20]. This paper helps to understand the evolution of ER by defining the four generations which address the problems of big-Data such as veracity, volume, variety, and velocity. The general ER solution design is proposed for each problem. In this thesis, we resolve the problem of variety and velocity.

A non-learning-based ER [EPSC19] method calculates the similarity between the attribute values using similarity metrics, and the entity pairs are classified based on the pre-defined rules and thresholds. However, to define the matching rules and thresholds, this method needs the involvement of domain experts. These issues are addressed by developing learning-based algorithms [RPHP17][INN08] to learn the linear functions and thresholds to classify the entity pairs. The learning-based ER methods adopt a classifier function such as SVM or BN based on handcrafted features from pre-labelled training entity pairs. However, these methods require feature engineering, and the outputs are not definable. Learning semantic and syntactic representation of entity descriptions is difficult for machine learning-based algorithms. As a result, several studies are being conducted on DL-based ER.

As discussed in Section 2.4, DL-based ER methods have shown promising results in learning the distributed representation of entity descriptions [NHH+19][MLR+18][ETJ+18]. The Deep Seq-to-Seq ER [NHH+19] and DeepMatcher [MLR+18] are compare-aggregate-based methods which consider tokens or phrases of entity descriptions to compare the multiple matching signals. These signals are then aggregated to decide the entity pair class. On the other side, the DeepER [ETJ+18] is a representation-based method that learns each entity's distributed representation, and then the similarity score is calculated using the representation vectors. The primary problems with these methods are: require a large training dataset, take more time to train the models from scratch or to update the existing model, and take more time to predict the class. In this thesis, we adopt the representation layers and classification layers from these studies. Since our thesis primarily aims to resolve the problem of velocity and variety, we do not consider the modules used in the above ER methods.

In Section 2.5, we discussed the pre-trained language model used to resolve the problem of ER. Since the PLMs are pre-trained on huge language models, these models achieve better results in representing entity descriptions. The representation vectors are then used to fine tune the task-specific layers. The papers Schema-agnostic-ER [TSS20] and DITTO [LLS+20] are discussed in the same section where they use a well-known pre-trained language model called BERT. Our thesis adopts a similar approach to model fine tuning as Schema-agnostic-ER and DITTO. The serialisation methodology discussed in Subsection 4.2.2 is adopted from the DITTO.

We adopt the CrowdER [WKFF12] based approach to get the true labels of entity pairs. This approach involves a human oracle to annotate the entity pairs. Our thesis system uses the approach defined in the paper [GH21] to get the continuous stream of entity pairs. This paper adopts incremental learning for blocking and produces the entity pairs in the stream for classification.

Table 2.1 provides an overview of all the related work. It is categorised on certain broad features that are important with respect to thesis implementation. In the table, the values of the learning-based algorithmic foundation represent as follows: ✓ indicates Machine Learning based algorithms, (✓) indicates Deep Learning-based algorithms, and (✓*) indicates pre-trained language models. In all other columns the ✓ represents that, the particular study uses the mentioned features.

Paper	Schema Awareness		Algorithmic Foundation		Incremental	Crows-Sourcing
	Schema Aware	Schema Agnostic	Non-Learning	Learning-based		
MinoanER [EPSC19]		✓	✓			
Gradient-based Matching [RPHP17]	✓			✓		
BN-based ER [INN08]		✓		✓	✓	
Deep Seq-2-Seq ER [NHH+19]	✓	✓		(✓)		
Deep Matcher [MLR+18]	✓			(✓)		
DeepER [ETJ+18]	✓			(✓)		
Schema-Agnostic ER [TSS20]	✓	✓		(✓*)		
DITTO [LLS+20]	✓	✓		(✓*)		
Task-based parallelization ER [GH20]	✓	✓	✓		✓	
Crowd-ER [WKFF12]	✓		✓			✓
Our System	✓	✓		(✓*)	✓	✓

Table 2.1: Comparison of Related Work of Entity Resolution.

3 Architecture Overview

Following a discussion of related work, in this chapter we discuss the end-to-end architecture of our DistilBERT-based ER system. Figure 3.1 depicts the architecture of our DistilBERT-based ER system. Initially, we present a high-level description of the first layer, *Prediction Layer*, where the trained model is used to forecast the streaming entity pairs. Next we go through a high-level overview of the second layer, *Training Layer*, where the DL-based model is trained iteratively. These two layers are independent of one another; that is, the prediction and training layers conduct their respective functions in parallel. The only connection between these two layers is $is\_model\_present$ flag, which is changed after the first iteration, and the matcher, which is updated with the most recent model in each iteration. The implementation of end-to-end system is explained in the Chapter 4.

3.1 The Prediction Layer

The prediction layer, seen in the orange box in Figure 3.1, comprises many components, such as Entity lookup table, Serialization, Cosine similarity function, Matcher, and Candidate pool.

The input stream is generated by the blocking stage of Gazzirri et al. [GH21], which consists of entity pairs with entity identifiers from both sources and weight indicating the similarity score determined in the *pairwise comparison* step. In the following components, the entity pairs are processed one at a time. The *entity lookup table* is used to get entity descriptions from JSON files. The JSON files are searched for entity identifiers from pairs, and the related description is retrieved.

The entity descriptions are sent as input to *serialization*, which is adopted from DITTO [LLS+20]. This method converts the input entity descriptions from JSON objects to a sequence of tokens, compatible to the DistilBERT's [SDCW19] input. Once the sequence of tokens has been produced, our system looks for the condition *is_model_present*, which looks for the fine-tuned DistilBERT-based classification model. If the model is available, the series is fed to the *matcher*; otherwise, to *cosine similarity function*. This condition is introduced since the fine-tuned model will not be present at first, i.e., is_model_present is FALSE. We employ the cosine similarity function, which is an external similarity function. The input stream is fed to the cosine similarity function, which delivers the similarity score for each entity pair description until the model is fine-tuned for the initial batch of data in the first iteration.

Once the first batch of data is trained in the first iteration, as described in the training layer, the fine-tuned model is deployed, and the *is_model_present* condition is set to TRUE. The input stream is now fed to the matcher, where the model predicts the class for input entity descriptions and offers the model's confidence. These results are sent to the analyst and saved in the *CP*, which serves as input to the training layer.

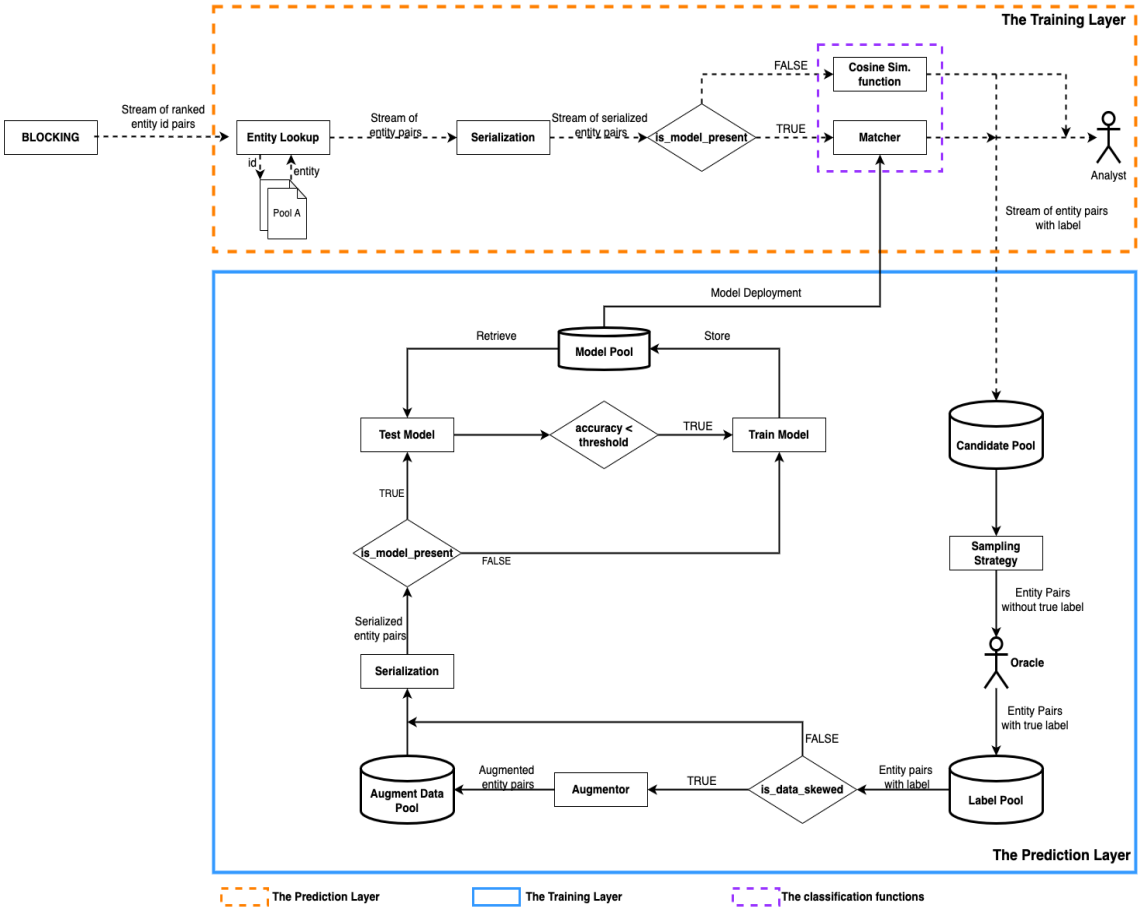


Figure 3.1: End-to-end architecture of Deep Learning in Stream Entity Resolution.

3.2 The Training Layer

The training layer is an iterative layer in which we examine a batch based on the timestamp in each iteration and then label, augment, and train the model using the data points from this batch. As a result, our model continues to learn data from incoming stream entity pairs. This layer is shown by the blue box in Figure 3.1, and it consists primarily of Data labeling, Augmentation, Serialization, Model training, and Model deployment processes. Below is a summary of all the operations that correspond to one iteration.

The *CP* is divided into two collections: primary and secondary. The output of the prediction layer, i.e., the stream of entity pairs together with predicted class label and confidence, is stored in the main collection, corresponding to continuous data storage. We employ a time-based sliding window to sample the batch of data from the continually growing collection. The window's distance matches the predetermined time in "minutes". All data points that appear inside this time frame are transferred to the secondary collection.

To get the true labels of the entity pairs, our system uses the *AL* and Crowd-sourcing-based approaches. An oracle (e.g., a human annotator) is used in this approach to annotate the entity pairs. The system obtains a batch with predefined batch size from CP's secondary collection using a predefined sampling strategies. We offer four techniques, which are detailed in the next chapter. An oracle annotates each entity pair one by one. Once the whole batch has been annotated with a true label, it is saved in the LP.

For the training of the DistilBERT-based model, the labelled entity pairs from *LP* serve as the training batch. The system calculates the proportion of match-entity pairs prior to the model training (in the ER problem, there are more entity pairs that do not match than entity pairs that do match). If this percentage is less than the predefined percentage threshold of augmented entity pairs, we use augmentation methods to increase the number of match-entity pairs. Otherwise, the batch is sent to the *serialization* phase. To accomplish augmentation, our system uses the NLPAUG [Ma19] library, which accepts a string as input and returns an augmented string as output. This library includes augmentation operations such as replacing a word with a synonym or antonym, keyboard distance error and replacing a character in a word with a special character, and many others. More information on the augmentation operations used by our system is covered in the following chapter. After performing the augmentation, the augmented entity pair batch is saved in the augment data pool.

The same *serialization* stated in the prediction layer is used in the training layer. The serialization takes the batch of entity pairs from LP as input and converts the entity descriptions from JSON objects to a sequence of tokens with special embedded tokens. The system then evaluates the condition $is\_model\_present$, which verifies whether the model is present in the model pool. The model will not be present during the first iteration, so the first batch is used as input to the model training phase, but the model will be present from the second iteration; thus, the second batch is used as input to the model testing step and then to model training.

We are motivated by DITTO [LLS+20], and Schema-agnostic-ER [TSS20] findings, which employ a pre-trained language model BERT followed by a fully connected layer and a softmax output layer for binary classification. Hence, in our work we use DistilBERT, which is distilled version of BERT model. *Model training* includes processes such as splitting the dataset for training and testing the model, and monitoring metrics such as F1-score, accuracy, and model errors. After training, the trained model is saved in the model pool and deployed as a matcher. The flag $is\_model\_present$ is then set to TRUE. As a result, the input stream of entity pairs is passed to matcher, which predicts the class label.

The model is evaluated on the input serialized batch beginning with the second iteration, prior to the model training phase. If the test accuracy falls below a pre-defined threshold, the model training phase is executed; otherwise, the batch is rejected, and the system carries on to the next iteration.

4 Concepts and Implementation

The previous chapter presented an overview of the architecture of this thesis. In this chapter, we discuss the implementation and the corresponding concepts supporting the implementation of those components in detail. The flow of this chapter is as follows. First, Section 4.1 describes the notations used in this chapter and the streaming input data. Section 4.2 discusses the implementation of the prediction layer, where implementation and concepts of a look-up table, serialisation, and similarity functions are discussed as subsections. Similarly, Section 4.3 discusses the Training layer, where implementation and concepts of data labelling, augmentation, model training and testing, and model deployment are discussed as subsections.

4.1 Notations and Stream Input

As an input, our thesis pipeline considers a stream of candidate tuple $c_{ab} = \langle a, b, w_{ab} \rangle$ where $a \in D_A$, $b \in D_B$ and w_{ab} is similarity score. a and b are identifiers of D_A and D_B which consists of finite entity descriptions (entity pools). An entity description e is a set of key-value pairs $e = \{(attr_i, val_i)\}_{1 \leq i \leq k}$, $attr_i$ is the attribute name, and val_i is the attribute's value. The entity pools collect structured and semi-structured data, such as JSON files. These pools contain heterogeneous data where each entity consists of the same or a different number of attribute-value pairs. Figure 4.1(a) shows that all three entity descriptions have different attribute-value pairs. We introduce entity tuples $e_{ab} = \langle e_a, e_b, w_{ab} \rangle$ where e_a and e_b are the entity descriptions of identifiers a and b, respectively. The output of our thesis pipeline is a stream of tuple $c^m_{ab} = \langle c_{ab}, l^m_{ab}, conf_{ab} \rangle$ where c_{ab} represents candidate tuple, $l^m_{ab} \in \{match, not - match\}$ defines the label predicted by matching function M, and $conf_{ab}$ defines the confidence of matching function.

Apache Kafka[1] is a distributed system consisting of servers and clients communicating via a high-performance Transmission Control Protocol (TCP) network protocol. It consists mainly of four components: Events, Topics, Producer, and Subscriber. Events are similar to messages which consist of primarily the key, data as value, timestamp, and optional metadata. These events are generated by the producer and sent to a topic, similar to a file system that stores and maintains the related events. The consumer subscribes to the topics to consume the events in the stream. Apache-Kafka is fault-tolerant, scalable, high-throughput, and independence between consumer and producer motivated us to use this in our thesis to consume the candidate tuples. In our thesis, we adopt apache-kafka to produce and consume the continuously streaming candidate tuples. The system subscribes to the specified topic and the corresponding events (candidate tuples) are consumed as input to our system. The configuration of Kafka is pre-defined in the configuration file of our system.

[1] Apache-Kafka Introduction: https://kafka.apache.org/intro

4.2 The Prediction Layer

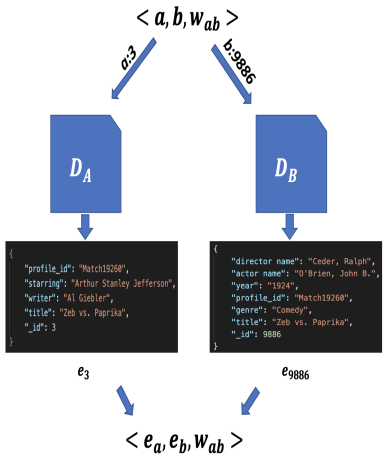
The prediction layer comprises components such as an entity look-up table, serialization, classification functions, and candidate pool. The streaming candidate tuples obtained from the apache-kafka are input to these components. The detailed implementations of these components are discussed in the subsequent subsections.

4.2.1 Entity Lookup Table

The first component of the prediction layer is the entity lookup table which is used to retrieve the entity descriptions e from entity pool D. This component takes entity identifiers a and b as input and outputs the entity tuple e_{ab}, where it consists of corresponding identifier's entity descriptions (e_a, e_b) and similarity score (w_{ab}).


```
[
    {
        "profile_id": "Match5316",
        "starring": "Gayne Whitman",
        "title": "The Silver Lining",
        "_id": 0
    },
    {
        "profile_id": "Match17874",
        "starring": "Karen Blanche Ziegler",
        "_id": 1
    },
    {
        "profile_id": "Match15948",
        "starring": "Mincha",
        "writer": "Sujatha",
        "title": "Kannathil Muthamittal",
        "_id": 2
    },
    {
        "profile_id": "Match19260",
        "starring": "Arthur Stanley Jefferson",
        "writer": "Al Giebler",
        "title": "Zeb vs. Paprika",
        "_id": 3
    }
]
```


(a) List of entity descriptions in D_A



(b) Entity description extraction.

Figure 4.1: Entity description extraction technique.

In this component, we implement the entity pools D_A and D_B as the list of dictionaries. Each entity's attribute-value pairs are mapped as a dictionary along with identifiers. Figure 4.1(a) shows an example of entity descriptions from the entity pool consisting IMDB dataset. Once the component receives the candidate tuple c_{ab}, it retrieves the identifiers, searches through the list of dictionaries and returns the entity description along with the identifier. An entity tuple e_{ab} is formed using these returned entity descriptions along with a similarity score (w_{ab}) representing the component's output. Figure 4.1(b) depicts the example of retrieval of entity description e. In this example, D_A refers to IMDB data source, and D_B refers to DBPedia data source. The candidate tuple c_{ab} consists of 3 and 9886 identifiers and the corresponding entity descriptions are shown.

4.2.2 Serialization

A pre-trained language model such as DistilBERT [DCLT18] takes a sequence of tokens as input, i.e., text. A key challenge is to transform the attribute-value pairs of entities into a sequence of tokens. We adopt the serialization method from DITTO [LLS+20]. DITTO serializes the entity descriptions as follows: for each entity description consisting of attribute-value pairs, i.e., $e = \{(attr_i, val_i)\}_{1 \leq i \leq k}$, it introduces a serialize function such that,

$$serialize(e) ::= [COL]\ attr_1\ [VAL]\ value_1,\ \ldots\ , [COL]\ attr_k\ [VAL] value_k$$

where $[COL]$ and $[VAL]$ are unique tokens indicating that following tokens are attribute names and attribute values respectively. To serialize an entity tuple (e_a, e_b), DITTO proposes serialize function below,

$$serialize(e) ::= [CLS]\ serialize(e_a)\ [SEP]\ serialize(e_b)\ [SEP]$$

where $[SEP]$ is the unique token that separates the two sequences and $[CLS]$ is the unique token required for DistilBERT to encode the sequence pair into a 768-dimensional vector that will be fed into the fully connected layers for classification.

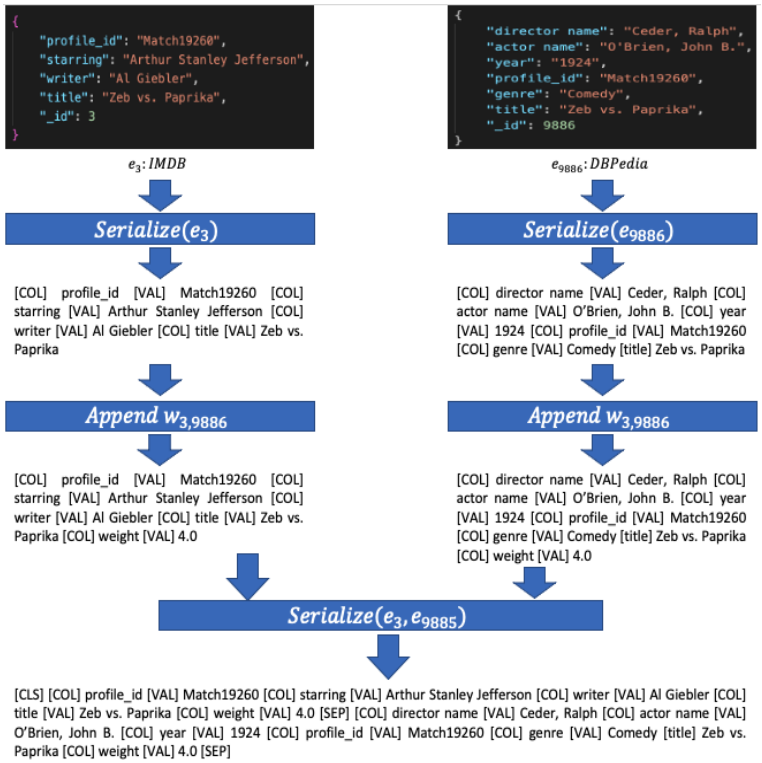


Figure 4.2: Example of serialization component.

Our thesis applies two changes to the DITTO's serialize methods: (1) It discards the identifier from the entity description, and (2) It appends the similarity score w_{ab} as a weight attribute at the end, i.e. $[COL]$ weight $[VAL]$ w_{ab}. Figure 4.2 shows the example of the serialisation approach. The example includes a pair of entity descriptions e_3 and e_{9886} from IMDB and DBPedia data sources respectively, where the number of attribute-value pairs varies. At first, the entity description serialize function is applied. The series is appended with similarity score w_{ab}, and at last, the entity pair serialize function is applied, including the unique token $[CLS]$ and $[SEP]$. The output of the serialisation component is input to the classification functions.

There are several methods for serializing entity descriptions in order for pre-trained language models to approach the input as sequence classification. For example, instead of concatenating the attribute names, concatenate just the attribute value as a series of tokens. This may be used when attribute names aren't relevant for classification or when the object only has one attribute. An extensive research on these serialization methods is left for future work.

4.2.3 Classification Functions

The layer's next component is the Classification functions, which is used to classify the input entity pairs as match or not-match, as well as the calculation of confidence of the associated class. This component includes the cosine similarity function as well as the DistilBERT-based matcher. This component accepts serialized entities $serialize(e_a, e_b)$ as input and outputs a labelled tuple c^m_{ab}. Before delving further into these functions, let's look at another component that comes before classification functions.

Before the classification layer, we add a flag $is\_model\_present$ that checks whether the model is present in the model pool; if it is, the flag is set to TRUE; otherwise, it is FALSE. Until the training layer completes an iteration and trains the DistilBERT-based classification model for the first batch, the flag is set to FALSE. Once the model is trained and stored in the model pool, the flag is always set to TRUE. The processes that occur in the training layer are detailed in the next section. If the flag is set to TRUE, the incoming stream of serialized entities are routed to the matcher function; otherwise, the stream is routed to the cosine similarity function.

Our thesis adopts two classification functions: (1) Cosine similarity function and (2) DistilBERT-based matcher. These two functions are discussed in detail below,

1. **Cosine similarity function**: This function computes the similarity among entity pairs by comparing the similarity between two vectors in an inner product space. It is calculated by taking the cosine of the angle between two vectors and determining if two vectors are heading in the same general direction. Because this function accepts vectors as input, we use TF-IDF vectorization, as mentioned below.

   The cosine similarity function accepts serialized entity descriptions as input and splits the input with regard to a special token $[SEP]$ to separate the two descriptions. The attribute and its values are concatenated after special tokens such as $[COL]$ and $[VAL]$ are removed. Following that, we use the TF-IDF algorithms [Joa96] to vectorize these two entity descriptions.

The Bag of Words (BoW) model underpins Term Frequency-Inverse Document Frequency (TF-IDF), which provides insights into the less important words in a document (entity description). The relevance of words is quite important in the text-similarity computation. The frequency of words (w) in a document (d) is measured by term frequency (Term Frequency (TF)). The TF definition is provided in the equation 4.1. Because the corpus documents are of various lengths, the denominator term in the equation is used to normalize. The first stage is to create a lexicon of unique terms and determine the TF for each text. TF will be higher for words that occur often in a text and lower for unusual ones. Because TF does not assess the value of a word, Inverse Document Frequency (IDF) is the measure of its importance. Each word in corpus D is given a weightage by IDF depending on its frequency. The equation 4.2 is used to define the IDF . A document's TF-IDF vector may be determined by multiplying the TF and IDF values, as illustrated in the equation 4.3.

$$TF(w,d) = \frac{occurrence\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d} \tag{4.1}$$

$$IDF(w,D) = \ln(\frac{total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w}) \tag{4.2}$$

$$TFIDF(w,d,D) = TF(w,d) * IDF(w,D) \tag{4.3}$$

Cosine similarity function [TJ13] is similarity function that is used to compare the documents. The equation 4.4 represents the definition of cosine function. In our work, the TF-IDF vector of e_a and e_b are represented as $\vec{x}$ and $\vec{y}$ in the equation 4.4.

$$similarity(x,y) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|\ \|\vec{x}\|} = \frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{4.4}$$

where $\|\vec{x}\|$ is the euclidean norm of vector $x = (x_1, \ldots, x_n)$, defined as the length of the vector. Table 4.1 and 4.2 represents the example of TF-IDF vectors of entities e_3 and r_{9886} respectively. These vectors are considered as $\vec{x}$ and $\vec{y}$ in the equation 4.4 to calculate the cosine similarity. The $similarity(e_3, e_{9886}) = 0.36$. The labelled tuple c_{ab}^{m} consists this score as confidence and label as None, then the tuple is stored in the CP.

words	title	zeb	vs	paprika	writer	al	gabriel
TF-IDF	1.41	1.41	1.41	1.0	1.0	1.0	1.0

Table 4.1: TF-IDF score of e_3.

words	title	zeb	vs	paprika	director	name	ceder	ralph
TF-IDF	1.41	1.41	1.41	1.41	1.0	1.0	1.0	1.0

Table 4.2: TF-IDF score of e_{9886}.

2. **Matcher**: Another classification function that we investigate in our study is the DistilBERT-based model (Matcher). The fine-tuned PLM is followed by task-specific layers in the matcher. The input sequence of tokens is encoded into machine-readable vectors using the fine-tuned DistilBERT model (fine-tuning of the DistilBERT model is explained in Subsection 4.3.4) and then the task-specific layers are used for classification. The input series contains a special token $[CLS]$ that indicates the classification task for the DistilBERT model, as explained in the serialization subsection. The 768-dimensional vector for each token represents the model's output. Out of all these vectors, we consider the vector corresponding to a special token $[CLS]$, because this vector comprises distributed representation of input entity descriptions.

   The task-specific layers use the $[CLS]$ token's 768-dimensional encoded vector as input. Because ER is a classification task, we use a fully connected Feed Forward Neural Network (FFNN) and a softmax layer for bianry classification. These layers' weights are learned via supervised training (explained in Subsection 4.3.4). The class of entity pairs is predicted using these weights. The process of entity pair classification is shown in the Figure 4.3.

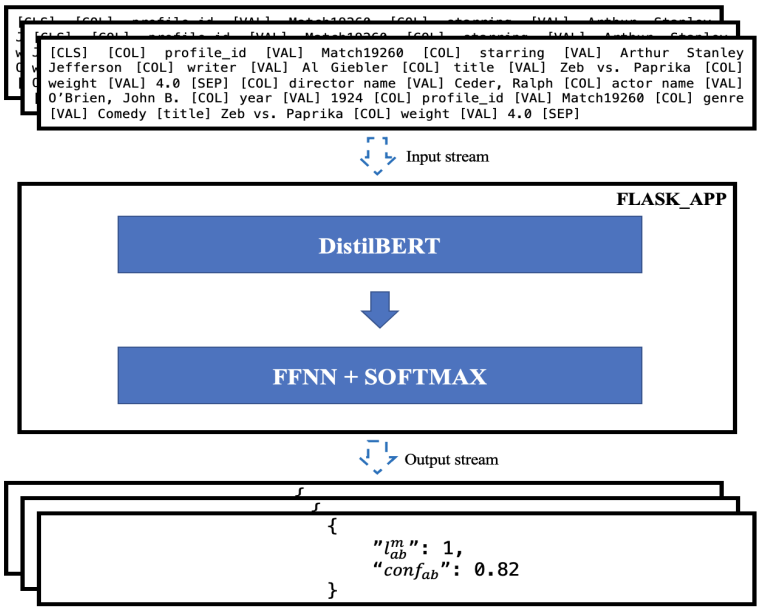


Figure 4.3: Flask based entity pairs classification.

The FLASK[2] framework is used to classify the streaming entity pairs. This framework is used to deploy the trained model. The predict function in the flask application employs a pre-trained model to categorize the input entity pairs. The streaming entity pairs are provided to the application through a request function. Using the predict function, the program predicts

[2]Flask: https://flask.palletsprojects.com/en/2.1.x/

the relevant class and calculates confidence. As an application response, the class and confidence are returned. A novel model deployment algorithm is presented to deploy the trained model (discussed in detail in Subsection 4.3.5). This approach combines two flask apps and a common variable to predict the class of streaming entity pairs without disrupting the stream, and the most recently updated model is used as a classifier. The flask_num variable contains an integer number (Zero, One, or Two) that indicates which flask application is active at the time. Zero value of flask_num indicates that no flask application is running hence the stream uses a cosine similarity function, one indicates the first flask application (flask_app1) is using the updated model hence, the stream uses flask_app1, and two indicates the second flask application (flask_app2) is running, hence the stream uses the flask_app2.

The similarity functions discussed in Subsection 4.2.3 produces a similarity score in the form of confidence. The cosine similarity function returns a confidence score for entity pairings but does not predict a class label. The DistilBERT-based classification function predicts the class and calculates the confidence of entity pairs. In the labelled tuple c^m_{ab}, the predicted class and confidence are entered. The labelled tuple is subsequently saved in the CP, which is fed into the training layer.

This completes the implementation of the prediction layer. During the implementation of components, mainly we encountered a couple of challenges which are listed below.

1. The **look-up table** was first developed using MongoDB. Two distinct collections in MongoDB are used to contain the entity descriptions from two separate data sources. The relevant entity description was looked up for each pair of constantly streaming entities. This required Input-Output (I/O) operations with respect to MongoDB for each entity pair, which prolonged the prediction layer's processing time. We implemented in-memory dictionaries to address this issue. The dictionaries are used to keep the entity descriptions from the two sources in the main memory. As a result, each entity pair's processing time is reduced. The drawback of this method is that the in-memory dictionaries cannot be utilized if the size of the entity descriptions exceeds the system's main memory size.

2. Using the learned classification function, continuously streaming entity pairs are categorized and then stored in the **Candidate Pool**. The next section, which makes use of MongoDB, provides an explanation of how the candidate pool is implemented. For each continuously streaming pair of classified entity pairs, similar to the prior challenge, the MongoDB insert operation is invoked. As a result of this procedure, there are more I/O operations, which increases storage time. This problem is addressed with the use of main memory and storing the multiple entity pairs at once. We use a list with a predetermined size. The list contains all continuously streaming pairs of classified entities. Once the list is full, we store it in MongoDB using the multiple data insert method. As a result, there are fewer I/O operations overall, improving storage time. The drawback of this strategy is identical to the drawback of the look-up table. The main memory's available free space imposes a restriction on the list's predetermined size.

4.3 The Training Layer

We reviewed the implementation of the prediction layer in the previous section, where the output labelled entity pair is stored in the CP. In this part, we will go through how to implement the following layer, the training layer, which is an iterative layer. This layer aims to iteratively train the model in order to keep it up to date with incoming stream data. As an output, this layer produces an updated model for each iteration.

The pipeline of the training layer includes components such as Sampling strategies for retrieving sample data for labelling, Data labelling entails a human oracle annotating the entity pair with true labels, Augmentation to handle the issue of class imbalance, Model training and testing, and Model deployment to classify the incoming entity pairs. In each iteration, these components are processed sequentially. The implementation of these components is discussed in the following subsections.

4.3.1 Candidate Pool

The CP is a storage component which stores the continuous stream of labelled tuples and acts as input to the training layer. We introduce two collections in the CP: Primary and Secondary. Primary collection continuously stores the streaming labelled tuples where each tuple is stored as a JSON object. The batch of data is selected from the continuously populating primary collection and stored in the secondary collection. This batch from the population is selected based on the moving time window. The time window is pre-defined in the configuration file which is defined in "minutes". The system time is used as the window start time in the first iteration, and the window end time is determined by adding the window time to the system time. The secondary collection is filled with all the data points whose timestamp occur between these start and end timings. The preceding iteration's end-time serves as the start time for the next iteration. The window's start and end times are then determined in the same way for subsequent iterations.

Because the incoming entity descriptions contain a heterogeneous schema, we required storage that could manage this kind of data. As a result, we chose MongoDB[3]. MongoDB is an open-source NoSQL database. The structured, semi-structured, and unstructured data can be stored in this. It can handle non-relational queries because it maintains a non-relational, document-oriented data model. MongoDB is very adaptable, allowing to aggregate and store many kinds of data. It employs the Binary JSON (BSON) document storage format, a binary JSON variant that can hold more data types. MongoDB stores each data as a document in a collection. Collections are like a table in a relational database which stores similar kinds of documents. Documents are made up of key-value pairs, which are the fundamental unit of data in MongoDB. The structure of the document can be modified by updating or adding the key-value pairs. As a unique identifier, documents may establish a primary key, and values can be any data type, including other documents, arrays, and arrays of documents. MongoDB uses MongoDB Query Language (MQL) to interact with the database, which functions similarly to Structured Query Language (SQL).

We use the PyMongo library which is a driver that interacts with MongoDB from python. Some of the important queries used in the systems are to insert and find the documents. The first query 4.1 shows the insertion of a labelled tuple. The second query 4.2 retrieves all the documents

[3]MongoDB: https://www.mongodb.com/what-is-mongodb

Listing 4.1 MongoDB query to insert classified tuple.

```
db.primary_collection.insert({
            _id: "3|9886|4.0",
            entity_a: {
                "profile_id": "Match19260",
                "starring": "Arthur Stanley Jefferson",
                "writer": "Al Giebler",
                "title": "Zeb vs. Paprika",
            },
            entity_b: {
                "director name": "Ceder, Ralph",
                "actor name": "O'Brien, John B.",
                "year": "1924",
                "profile_id": "Match19260",
                "genre": "Comedy",
                "title": "Zeb vs. Paprika (1924)",
            },
            label_m: null,
            confidence: 0.36
        });
```

Listing 4.2 MongoDB query to retrieve entities from CP's primary collection.

```
db.primary_collection.find({
    timestamp: {
        '$gt': 1655570533,
        '$lte': 1655570653
    }
});
```

present between 18-06-2022 18:43:00 and 18-06-2022 20:43:00. The numbers represent the unix timestamp. These retrieved documents are stored in the secondary collection.

4.3.2 Data Labelling

Data Labeling is the next step in the pipeline. This component seeks to overcome the issue of the huge annotated data set being required to train the model. The component obtains labelled tuples from CP and annotates them with true labels in our work. The true labelled tuples are the component's output. Human involvement with the system is expensive and time-consuming, but it is more precise in the ER [WKFF12]. To annotate the entity pairings, we use crowd-sourcing-based ER [WKFF12] and AL [Set09], where each entity pair is displayed to a human oracle and requested to annotate the data. The annotated data, which consists of oracle labelled tuples $c^o_{ab} = \langle c_{ab}, l^m_{ab}, conf_{ab} \rangle$, is stored in the LP, where the annotated tuple l^o_{ab} substitutes the predicted label l^m_{ab} in a labelled tuple. In Section 2.6.2, we went through the crowd-sourcing-based ER in-depth and in the next paragraph we discuss AL in detail. The concepts and implementation of data labeling are also discussed below.

Active Learning (AL) is a kind of machine learning in which a learning algorithm interacts with a person to classify new data points with intended outcomes. It is employed in cases where the unlabelled data set is vast and the supervised model's accuracy may be improved by prioritizing annotation. The Figure 4.4 depicts the general phases involved in AL. The first phase, *Active Query Selection*, involves manually labelling a sub-sample of the unlabelled data. The next phase is for a *human oracle* to annotate each data point. A human oracle is represented by the brain in the Figure 4.4. The supervised model is trained using the labelled data set, and then it is used to predict the class for each streaming data set. This iteration continues to improve the accuracy of the model over time.

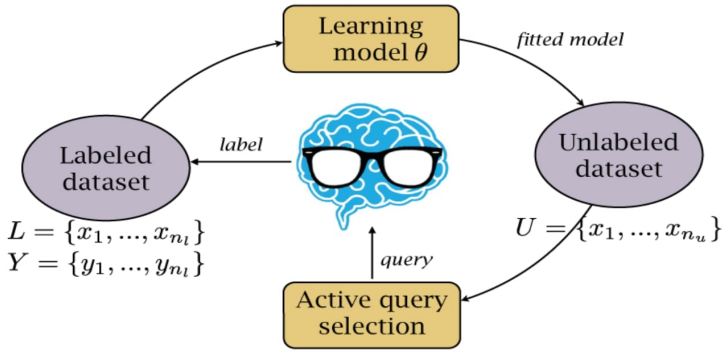


Figure 4.4: Active learning process[4].

We leverage the weight w_{ab} of the entity pairs and the confidence of the model's prediction to add four strategies as active query selection in our pipeline. The sub-sample data set is retrieved from CP's secondary pool using these strategies. The strategies are applied to the secondary pool, and then a sub-sample of the pre-defined number of data points N is obtained. The following are the sub-sampling strategies:

- **Strategy 0 (Default)**: The labelled entity tuples c^m_{ab} are sorted in decreasing order based on the entity pair's weight w_{ab} and model confidence confab in this technique. The sub-sample of size N is then obtained for labelling from the population of CP's secondary collection.
- **Strategy 1**: The labelled entity tuples c^m_{ab} are sorted in decreasing order depending on the model's confidence in this technique. The first N data points from the CP's secondary collection are then extracted and labelled.
- **Strategy 2**: This technique involves sorting the labelled entity tuples c^m_{ab} in decreasing order depending on the model's confidence, and then collecting the first $\frac{N}{2}$ and final $\frac{N}{2}$ data points as a sub-sample for data labelling.
- **Strategy 3**: The labelled entity tuples c^m_{ab} are sorted in decreasing order depending on the model's confidence in this technique. The data set in CP's secondary collection is divided into three bins depending on tuple order: High, Medium, and Low bins. Consider the sorted

[4]What is Active Learning?: https://deepai.org/machine-learning-glossary-and-terms/active-learning

secondary pool with a population (n). First, $\frac{n}{3}$ data points are categorized as high confidence bin, next $\frac{n}{3}$ data points are categorized as medium confidence bin, and finally, the remaining data points are categorized as low confidence bin. The user picks the appropriate number of data points from each bin for labelling, which adds up to N, after pre-defining the percentage selection for each bin.

The recovered sub-sample is annotated by a human oracle. In our workflow, each data point is given to a person who is then requested to annotate the data. The annotated data point is then placed in the LP as a true labelled tuple c^o_{ab}. We created a script that annotates the entity pairings for the sake of our experiment. The entity pairs and true labels were included in the data set we used for our research. The MongoDB database *Ground-Truth collection* stores these true labels. In a loop, the entity pair identifiers a and b are picked from the sub-sample, and then the ground-truth collection is searched for these identifiers. If the identifiers are present in the collection, the entity pairs are marked as $Match$; otherwise, they are marked as $Not - Match$. In the labelled tuple, the oracle annotated label l^o_{ab} is changed, and c^o_{ab} represents the new tuple. The LP, which is also implemented by the MongoDB database's collection, then stores this tuple.

4.3.3 Augmentation

Before we go into the augmentation, we will have a look at another flag called is_data_skewed, which is checked before the augmentation is performed. The percentage of match entity pairs is determined once the true labelled tuples c^o_{ab} are stored in the LP. This is accomplished by computing the percentage of labelled tuples c^o_{ab} with true label l^o_{ab} as one. If this percentage is less than the pre-defined augmentation percentage (aug_perc), the data set is skewed, and the flag is set to $TRUE$; otherwise, the flag is set to $FALSE$. The augmentation procedure is performed to the data set if the flag is $TRUE$. Otherwise, the data set is sent to the serialization stage.

The augmentation component comes next in the pipeline. This component takes a sub-sample of labelled tuples c^o_{ab} as input and returns a sub-sample of labelled tuples with augmented data points as output. This component is designed to address the issue of an unbalanced data set. It is an unbalanced data set if the data set is biased towards one class. Primarily, there are three approaches to resolving this problem; *oversampling* where the number of smaller class data points is increased, *undersampling* where the number of data points is decreased, and *augmentation* where the errors in the data set are added manually.

By integrating all three methodologies described above, this thesis presents a novel strategy to deal with the issue of unbalanced data sets. This method is described in detail below:

- From the sub-sample of LP, the total number of match and non-match entity pairs is determined.
- The match entity pairs are over sampled by applying augmentation to each match entity pair, where the number of match entity pairs is doubled.
- The match entity pairs are again over sampled by duplicating one entity description as another entity, i.e., consider a tuple $\langle e_3,\ e_{9886},\ 1,\ 0.36\rangle$ from a sub-sample of a labelled tuple c^o_{ab}, once the one entity is duplicated as another, the tuple looks like $\langle e_3,\ e_3,\ 1,\ null\rangle$. The augmentation is applied to the duplicate tuple. Only the entity pairs from the not-match class are considered for this approach.

- We determine the total number of augmented match-entity pairs only after performing all of the aforementioned stages, and then under-sample those number of pairs from not-match entity pairs.

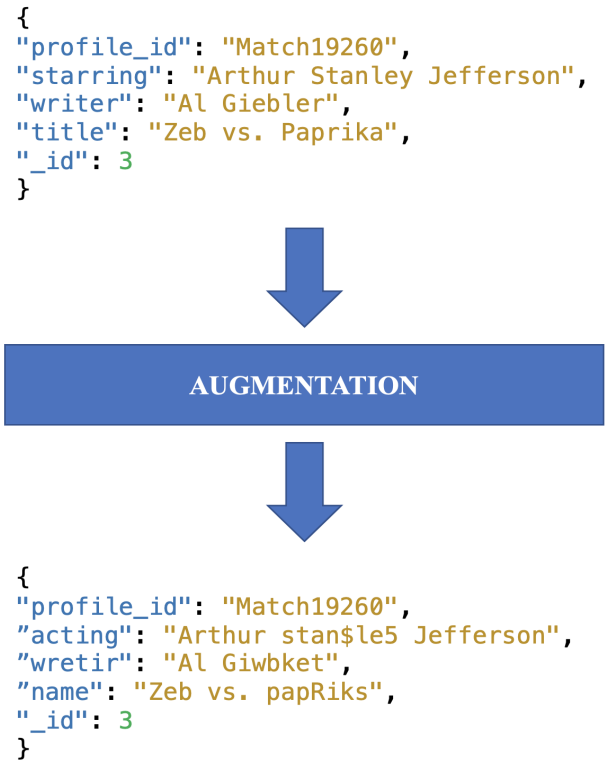


Figure 4.5: Augmentation applied on the entity description e_3.

We are using the NLPAUG [Ma19] open-source python package for the augmentation. It employs augmentation on both a character and a word level. Our thesis adopts *word* and *character level* augmentation which involves the *key-board error*, *random error*, *synonym change*, and *spelling mistake* operations. These operations are applied on both attribute names and values. We loop over each entity's key-value pairs, i.e., the attribute name and value. We have given the user the choice of selecting all operations or individual operations, word-level or character-level or both levels, and attribute name or attribute value or both for entity descriptions in our work. Before we execute our system, these parameters serve as configurations. The four primary augmentation operations listed are discussed in detail below,

- **Synonym Change**: This is a word-level operation where synonyms replace the words in the string. Consider the example in Figure 4.5, where the entity description e_3 is shown before and after applying the augmentation. The words "starring" and "title" were replaced with synonyms "acting" and "name", respectively.
- **Spelling Mistake**: This is a word-level operation where randomly misspelled words replace the existing words. Considering the same Figure, the word "writer" is misspelled to "wretir".
- **Key-Board error**: This is a character-level process that considers the characters of words for augmentation and substitutes randomly chosen characters with other characters depending on keyboard distance. Figure 4.5 shows the word "Giebler"being replaced with "Giwbierüsing

randomly picked characters. We can see that the letters *e* and *w* are next to each other in this example. Likewise, the letters *l* and *i*. Errors from the QWERTY keyboard are taken into account by our system.

- **Random error**: This is the character-level procedure in which key-board mistakes, special characters, and letter case changes are used to replace the word's characters. Consider again the same Figure, in the word "Stanley," the letter case is altered for the letter 'S', the character 'y' is converted to the number '5', and the unique character '$' is added. As a result, the word "Stanley" became "stan$le5". In the same way, the term "paprika" has been augmented to "papRiks".

The augmented entity pairs and non-augmented entity pairs are placed in the *augment data pool* once all of these actions have been applied to the entity descriptions e. The augment pool is implemented using the MongoDB database. The augment pool's entity pairs are used as input for the following component, serialisation.

4.3.4 Model Fine-tuning

The concept and implementation of our work's key component, fine-tuning model, is discussed in this section. The main aim of this component is to train the model over iterations. The components takes serialized tokens as input and produces a trained model as output. The trained model is then used in the model deployment component which is discussed in the next subsection.

Two components that are before the model training are serialisation and the $is\_model\_present$ flag. Serialization is implemented in the same way as explained in Subsection 4.2.2, where it is used to convert input entity descriptions from key-value pairs to a sequence of tokens. The condition $is\_model\_present$ checks whether the trained model in the model pool is present. The input data is passed to model testing if the model is present, otherwise to model training.

In our work, we employ the DistilBERT language model followed by a task specific layers as a model. DistilBERT is distilled version of BERT [DCLT18]. Since entity resolution is a classification task, we use FFNN and a binary softmax layers as a task specific layer. The series of tokens are embedded to machine-readable vector by DistilBERT model and then these vectors are trained on FFNN and softmax layers. Below we discuss concepts and implementation of these layers in detail.

Bidirectional Encoder Representations from Transformers (BERT)

BERT [DCLT18] is an universal language model that is pre-trained over large text corpora such as wikipedia articles. Bidirectional Encoder Representations from Transformers (BERT) is a term that describes the importance of bidirectional learning. The transformer encoder stack in the BERT model is learned during the pre-training. There are multiple versions of BERT, some of them are, BERT-Base (12 layers), BERT-Large (24 layers), and DistilBERT (6 layers). The series of token embedded with special tokens is provided as input to this model, where the number of tokens are limited to 512. Then, it produces a 768-dimensional embedded vector for each token. The output

vector of token $[CLS]$ comprises the distributed representation of the input tokens. This vector is produced using the layers of encoders present in the BERT model. The pre-training procedure involves two tasks which are listed below,

- **Token Masking**: This task involves the bidirectional model training, i.e., the model is trained from both left to write and right to left of the input. In order to achieve the bidirectional training, 15% of input tokens are masked and then predict the masked tokens. The corresponding final vectors of masked tokens are fed to a softmax layer over the vocabulary.
- **Next sentence prediction**: This task aims to tackle the problem of understanding the relationship between sentences. This task is used in the natural language inference and question answering. In order to understand the relationship between the sentences, the model is fed two sentences and asked to predict if the second sentence follows the first sentence.

DistilBERT

DistilBERT [SDCW19] is distilled version of BERT, which makes it lighter, cheaper, faster and smaller. Distillation is a approach that is used to compress the large model (teacher) into a small model (student). DistilBERT uses a knowledge distillation (compression technique where the student is trained to imitate the behaviour of teacher) to reduce the model size to solve the problem of exponentially increasing transformer models cost. The toke type embedding and the pooler (used for next sentence prediction in BERT) layers are removed to reduce the model size by a factor of 2. The DistilBERT-based classifier architecture is shown in the Figure 4.6. The DistilBERT paper shows that the model achieves 97% of BERT model's performance with 40% reduced size. DistilBERT introduces a triple loss to pre-train the model viz, masked language modeling, distillation, and cosine embedding losses. The masked language modeling loss is explained above. In the distillation loss, the DistilBERT is trained on the BERT model's soft target probabilities. At last in the cosine embedding loss is used to align the directions of the DistilBERT and BERT hidden state vectors. This pre-trained model can be used for multiple tasks such as sentence pair classification, single sentence classification, question answering, and single sentence tagging.

In our thesis, we aim at solving the problem of classifying the streaming entity pairs. This requires a faster classifier to avoid the problem of bottleneck in prediction of entity pairs. According to the authors of DistilBERT, the model is 60% smaller and faster which makes it an appropriate embedding model compared to BERT, GPT-2, and Bidirectional-LSTM. We use the sentence pair classification task of DistilBERT as an entity pair classification.

To classify the entity pairs, a task-specific layers are added after the final layer of DistilBERT. In our work, we add a fully connected FFNN and a softmax output layer for binary classification. Since the output of last layer of DistilBERT model is a 768-dimensional vector, the size of the FFNN is 768. As an optimisation function, we employ AdamW [LH17], which decreases the loss and updates the parameters while model training phase. AdamW is a stochastic optimisation approach that decouples the weight decay function from the gradient update and is based on Adam [KB14].

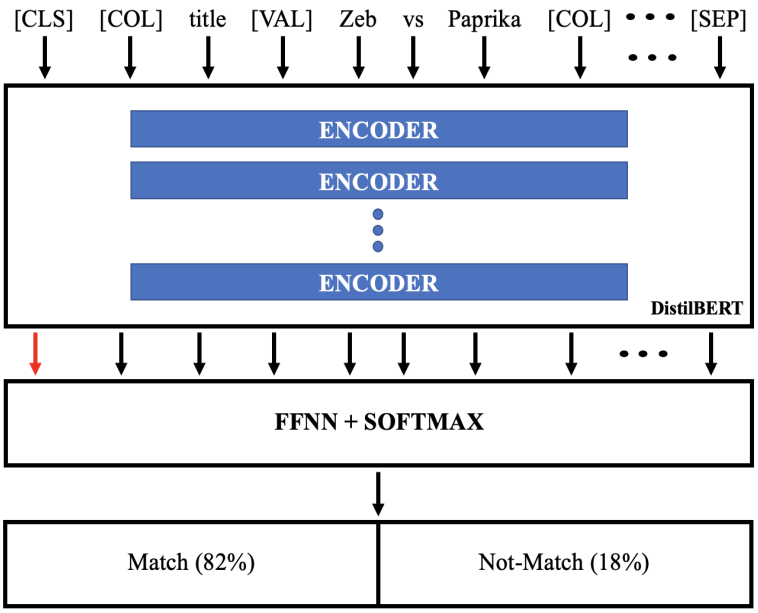


Figure 4.6: Components involved in the DistilBERT-based classifier.

Model Training

A trained model is not available in the model pool in the first iteration. As a result, the flag is_model_present is set to FALSE. Model training step receives the serialized data set and divides it into training, testing, and validation data sets (we adopt hyper-parameter from DITTO [LLS+20] except epochs and batch size). The training data set is used to train the model, the validation data set is used to determine the threshold for categorizing data points as $Match$ or $Not - match$, and the test data set is used to predict the labels on the trained model. If the model's validation f1-score is larger than the previous epoch's validation f1-score, the model is saved in each epoch of model training. The model and optimizer's state_dict (a state_dict is a model object that contains learnable parameter and hyper-parameter information) is stored in a model.pt file. In addition to state_dict, the model.pt saves the best validation f1-score, best test f1-score, and probability threshold in each iteration. The model.pt is used in the model deployment as a matcher.

Model Testing

The flag is_model_present is always set to TRUE starting with the second iteration. As a result, the serialized data set is fed to model testing. *Model testing* retrieves the stored model (model.pt) from the model storage, as well as the state_dict and other parameters from the model.pt file. The preceding iteration's state_dict and probability threshold are used to test the model, and the appropriate test loss is computed. The data set is deleted if the loss is less than the pre-defined threshold, and the next cycle begins. Otherwise, if the loss exceeds a certain threshold, the model must be retrained using the fresh data set. Hence, the previously saved model's state_dict, optimizer's state_dict, and probability threshold are retrieved from the model.pt. These parameters

are used to train the model as before. To identify the streaming entity pairs, the updated model (model.pt) is saved in the model storage and deployed via flask. In the next section, we'll explore the implementation of model deployment.

4.3.5 Model Deployment

After the model has been trained, it needs to be deployed in a location to make it accessible to input data and obtain predictions. In this work, the FLASK[5] framework is used to deploy the model on the local host. FLASK is a python-based web application framework. It enables end-users to label entity pairs by interacting with the trained model. The entity pairs are sent to the flask application by the end-user as a request. The application uses the data and responses back with the corresponding entity pair's class.

We provide a flask-based model deployment approach that does not interrupt the classification of streaming entity pairs in this thesis. Below is a description of the algorithm. Prior to that, we'll go through the parameters and functions that were utilized to create this algorithm.

Two flask applications (flask_app1 and flask_app2) are considered to deploy the trained model. Each flask app has a *predict* function that use the trained model stored in the flask app storage (storage1or storage2). The *predict* function accepts an entity pair as an input and returns the predicted class as an output. We have built shell scripts that execute the commands as background processes to start and terminate each flask application. To keep track of which program is running at any given time, a shared variable (flask_num) is employed. Because the trained model is not there at the start, storage1 and storage2 are empty. Flask_num, a shared variable, is set to zero.

The method is broken down into several phases, which are detailed below,

- **Step1**: The model is trained and saved at the model pool in the first iteration, as mentioned in the previous subsection. After that, the saved model is replicated to storage1 and storage2. The model is used in both flask apps to predict entity pairs. The start shell scripts for both apps, which operate as background processes, are then executed. The shared variable is then changed to one.
- **Step2**: The model is trained and saved in the model pool in the following iteration, as mentioned in the previous subsection. The stop shell script is used to terminate flask_app2. The model is then moved to storage2 and the shell script is used to start flask_app2. The flask_num variable is then set to two, and the streaming entity pairs are categorised using the new model.
- **Step3**: The trained model is saved in the model storage again in the following iteration. Because the input stream is now categorized by flask_app2, flask app1 is now terminated using the stop shell script. The stored model is replicated to storage1. The start shell script is then used to start flask_app1. The variable flask_num is now set to one, indicating that the model has been updated.
- **Step4**: Step2 and Step3 will alternatively continue in the following iterations.

[5]Flask: https://flask.palletsprojects.com/en/2.1.x/

This algorithm does not provide an optimal solution since both the applications run always. The optimum solution for this problem would be to stop one application after starting another application. This solution is not explored in this thesis but could be taken as part of future work.

Implementation of model deployment completes the training layer's implementation. Below, we list a couple of the challenges faced during the implementation of components of the training layer.

1. The labelling script sequentially annotates entity pairings in the **Data Labelling** component. At first, when the script annotates the data, our system immediately stores the entity pairs. This increases the I/O operation for MongoDB (LP), as discussed in the challenges of the prediction layer. Therefore, to resolve this difficulty, we used the same technique that was utilized to resolve the issue while storing the classified entity pairs in the candidate pool, i.e., adopting lists. The limitation of this approach is the same as the limitation for the CP.
2. To **deploy the trained model** and classify the continually streaming entity pair data, we initially utilized a single flask application. Following each iteration's model training, the same kafka topic that serves as the training layer's input receives the poison pill. The consumption of streaming entity pairs is stopped as soon as the training layer receives this pill. The flask application then updates the training model. The input stream is started upon the updating of the model in the flask application. As a result, the continuous streaming entity pair classification has to be interrupted. To solve this issue, we introduced an algorithm that continuously updates the trained model in the flask without disturbing the continuous streaming entity pairs. Subsection 4.3.5 provides an explanation of how this algorithm is implemented.

5 Evaluation

This chapter details the experiments that were conducted to evaluate the performance of our end-to-end DistilBERT-based ER system implemented in this thesis. In the first step of the evaluation procedure, the characteristics of datasets used for the experiments are discussed. Then we set evaluation goals. This evaluation is designed to address these goals.

5.1 Dataset Characteristics

The properties of the input dataset are shown in Table 5.1. These datasets were taken from the JedAI[1] dataset library. Many components of ER literature, including Task-based parallelization ER [69], DeepMatcher [9], and DeepER [19], rely primarily on these. We have taken into consideration clean-clean ER data sets since the dirty ER does not exhibit heterogeneity. The input entity pairs for comparison are taken into consideration from two data sources, as stated in Section 3.1. The third column lists the data sources. The dataset for the Movie include details like actor and director names and movie titles. The full-text articles and bibliographic literature are described by the organizations that are a part of the DBLP, ACM, and Google scholar. The largest dataset used in our experiments is the DBPedia dataset. It includes objects from a variety of domains, including those for products, citations, and movies.

Information like the author, the paper title, and the publication year is found in the entities of the dataset DBLP, ACM, and Google Scholar. For DL-based ER problems, this information serves as input features. From the results of experiments, we can observe that the accuracy of a model is high since the majority of the features in these databases have the same characteristics. The feature names for the Movies dataset, however, differ according to the sources (shown in Figure 4.1a). As a result, in the majority of the experiments, we utilized the Movies dataset to assess our model.

Dataset	**Domain**	**Source**		**Size**	
		DatasetA	**DatasetB**	**DatasetA**	**DatasetB**
Movies	Movies	IMDB	DBPedia	27614	23181
DBLP-ACM	Citations	DBLP	ACM	2615	2293
DBLP-SCHOLAR	Citations	DBLP	Google Scholar	2615	61352
DBPedia	Hybrid	DBPedia1	DBPedia2	1.19M	2.16M

Table 5.1: Characteristics of input datasets.

[1] JedAI Toolkit: https://github.com/scify/JedAIToolkit/tree/master/data

5.2 Goals

The goals for this study are as listed below. They focus on evaluating our system based on the number of iterations of training, the data skewness, and the time taken to execute the components.

- To determine the quality of model learning over the number of iterations. The model quality is defined by the model's f1-score, train loss, and test loss.
- To determine the number of iterations the model takes to progressively converge to optimum point. The optimum point is determined by measuring the quality of the model in each iteration. The model is optimum if the quality of the model is greater than the threshold.
- To determine the performance of each component and overall system by measuring the execution time taken.

5.3 Design

In order to achieve the goals described in above section, certain hypotheses were defined based on the research objectives of this thesis, findings from related work, and our understanding of the entity resolution methods. In this section, we discuss the defined hypothesis and different conditions.

5.3.1 Hypotheses

- **H1**: The model quality improves over the iterations if the input data is from the same domain.
- **H2**: The model quality improves over time and takes a smaller number of iterations to converge to near-optimum performance if the upcoming data set is balanced with respect to matches and not-matches.
- **H3**: The model quality improves over iterations for the large training dataset.
- **H4**: The model quality varies over iterations for multiple sampling strategies to fetch the data set for labelling.

5.3.2 Conditions

To assess the model's performance and quality in terms of Data dkewness, Input domain, Training sample size, and Sampling techniques, nine conditions were developed. All of these conditions are summarized in Table 5.2. To verify the hypotheses, experiments are conducted under multiple conditions.

The table demonstrates that different conditions are created by altering the data origin domain, sample size, data skewness, and sampling techniques. Each of these conditions offers information to support one or more hypotheses (mentioned in Column two). The next paragraphs describe the significance of each parameter's value.

Condition	Hypothesis	Data origin Domain	Sample Size	Data Skewness	Sampling Strategies
C1	H1, H2	Same	Small	Medium	S3
C2	H1, H3, H4	Same	Medium	Medium	S3
C3	H3	Same	Large	Medium	S3
C4	H2	Same	Small	Low	S3
C5	H2	Same	Small	High	S3
C6	H4	Same	Medium	Medium	S0
C7	H4	Same	Medium	Medium	S1
C8	H4	Same	Medium	Medium	S2
C9	H1	Hybrid	Medium	Medium	S3

Table 5.2: Different conditions to evaluate the iterative model.

An important parameter that identifies the domain to which the entity pairs belong is the data origin domain. The Same, Hybrid, and Sequence domains comprise this parameter. The term "same domain" refers to the domain from which the streaming entity pairs were generated, i.e., all entity pairs belong to the domains of "movies" or "citations". The term "hybrid domain" denotes the streaming entity pairings generated randomly from multiple domains, including those for movies, citations, and products. The sequence domain denotes the entity pairs are successively generated from the domains of movies and citations.

The following parameter sample size describes the model's training sample size. As a training batch, the values small, medium, and large sizes consist of 5000, 10000, and 25000 data points, respectively. The bias toward a class is represented by the data skewness parameter. The labelled data in the low and high values includes data points that are 50% and 5% match, respectively, while the medium value has entity pairs that are 10% and 20% match entity pairs. Sampling strategies are the final parameter, and they are used to obtain the sub-samples from the candidate pool for annotation. In Subsection 4.3.2, the details of these strategies are covered.

When the streaming candidate tuples originate from the same or other domains, conditions C1, C9, and C10 evaluate the model quality. The effect of the input data domain on the model quality is evaluated by changing the data origin domain while maintaining all other parameters constant. The affect of training data on model quality and execution time is discovered in conditions C1, C2, and C3 by altering the sample size while keeping the other parameters constant. Similar to this, conditions C1, C4, and C5 as well as conditions C6, C7, and C8 are used to assess the impact of data skewness and sampling techniques on the model quality, respectively. These conditions assist in generating statistically significant evidence to support hypotheses (H1 to H4).

5.4 Performance Evaluation Metrics

To evaluate the performance of the developed system, a list of Independent and Dependent variables are defined. Independent variables are varied throughout the user study, and the effect of these variations on the dependent variables is measured. This in turn addresses the defined hypothesis and achieves user study goals. In addition to the independent variables, there are a few more variables called extraneous variables that are described in a different section.

5.4.1 Independent Variables

Independent variables are the cause for change in the performance of the system. Such causes are independent of each other and affect the dependent variables. Conditions defined in Subsection 5.2 incorporate most of these independent variables and demonstrate their variations across the tasks. This section explains them in detail,

1. **Datasets**: This is the most important independent variable. The datasets we consider for experiments allow us to evaluate the model for all hypotheses (H1-H4). The details of the data set used, and their characters are explained in Section 5.1.
2. **Sample Size**: The sample size variable represents the multiple batch sizes for evaluating the model quality and system's execution time (H3).
3. **Data Skewness**: The values of this parameter indicated the bias towards a class in a labelled dataset: This variable is used to observe the change in model quality.
4. **Sampling Strategies**: The variable represents different strategies used in the sub-sampling of the data to annotate from an oracle. The change in this variable corresponds to a shift in model quality.

5.4.2 Dependent Variables

Several dependent variables are defined and analysed to measure the effect of independent variables on the system performance. They are targeted to measure the system's performance with respect to the time and accuracy of the DistilBERT-based ER model. In every iteration of model training, we evaluate the below dependent variables by varying the independent variables.

1. **Recall (R)**: It is the fraction of relevant instances that were predicted. The effect of different independent variables on this variable is measured in terms of percentage in every iteration. This variable helps to evaluate the model for all hypotheses.
2. **Precision (P)**: It is the fraction of relevant instances among the predicted instances. Like recall, this variable is measured in terms of percentage in every iteration with respect to varying independent variables.
3. **F1-Score**: It sums up the model's predictive performance by combining the recall and precision. The F1-score is defined as $\frac{2PR}{(P+R)}$.

4. **Train Loss**: The metric explains how the model fits the training data sets. The cross entropy loss[2] function is used to calculate the loss.
5. **Test Loss**: The metric explains how the trained model predicts the test data set. Similar to train loss, the cross entropy loss function is used.
6. **Time**: CPU and GPU execution time of the end-to-end system and components of the system.
7. **Iteration**: A complete cycle of the training layer.

5.4.3 Extraneous Variables

Extraneous variables are the ones that have the potential to affect the outcome of the study but are not being investigated in the evaluation. Below is the list of extraneous variables for this evaluation.

Human Oracle: To annotate the data, we take into account a human oracle. Due to time restrictions, we used a python script as a replacement of the human oracle in our thesis work. The script annotates the entity pairs using the ground truth label file from the dataset. The attention and familiarity of the human oracle are discussed below.

Because the human oracle is skilled at entity resolution, it follows that it is also knowledgeable about entity descriptions. The entity description's columns may have special meaning, which the oracle may be aware of and use this skill to annotate the data. This might result in a lot of perfectly annotated data. However, an oracle that is unfamiliar with ER may wrongly annotate the entity pairings. Therefore, we presume that the entity pairs that the oracle annotates are always accurate.

Another significant factor in our research is the human oracle's level of attention. The attention and fatigue of the oracle affect the accuracy of the annotation, which may not seem like a big issue. If the system uses a human oracle, these superfluous factors should be taken into account in future work.

5.5 Experimental Setup

This section discusses the information required for executing the experiments to evaluate the model. We describe the parameters, hardware and software used to conduct the experiments.

5.5.1 Parameters

Below, we list the parameters used in our experiments. These parameters can be altered through the config file. All the model hyperparameters are adopted from DITTO [LLS+20].

1. **aug_flag**: This parameter controls the augmentation process by setting it to TRUE or FALSE.

[2]Cross Entropy: https://en.wikipedia.org/wiki/Cross_entropy

2. **th_match_perc**: This parameter represents the threshold which determines whether the augmentation is required for the training data set or not. The augmentation is applied to the training dataset if the percentage of match entity pairs in the LP falls below the threshold. This parameter has values of 5% and 10%.
3. **th_test_loss**: This parameter represents the threshold which decides whether the trained model's fine tuning is required. The test loss is calculated for the new iteration's data set; if the loss is more than the 5% threshold loss, then the pre-trained model is fine-tuned. Otherwise, the dataset is discarded. The value of this parameter is 0.05
4. **aug_key_val**: This parameter defines the key-value level augmentation of entity descriptions. If the parameter's value is set to 'KEY', then the augmentation is applied to the keys of entity descriptions. Similarly, 'VAL' defines the values of entity descriptions, and 'BOTH' determines the augmentation is applied to both keys and values of descriptions.
5. **aug_word_type**: This parameter represents the augmentation techniques used for entity descriptions at the word level. The values 'SYN', 'SPL', and 'BOTH' define synonyms and spelling, respectively.
6. **aug_char_type**: Like the above parameter, this parameter represents the augmentation techniques applied to entity descriptions at the character level. The values, 'SYN', 'SPL', and 'BOTH', define synonyms, spelling, and both, respectively.
7. **time_window_min**: This parameter depicts the time in minutes for moving window as discussed in the Subsection 4.3.1. All the predicted entity pairs having timestamps within this window are transferred to a secondary collection from a primary collection of CP.

5.5.2 Experimental Environment

To evaluate our end-to-end DistilBERT based ER model, we adopt a virtual machine. The hardware configuration is explained in the first subsection, whereas the frameworks and libraries used are explained in the second subsection.

Machine Configuration

In this thesis, we use the remote machine, which is connected through the secured shell protocol (SSH)[3]. The system's hardware configurations are listed below,

- **Operating System**: Ubuntu 20.04
- **Processor**: 2 x AMD EPYC 7763, 2.45GHz, 64 Cores, 128 Threads
- **Memory**: 1TiB
- **Graphics**: 4 x NVIDIA A100, 40GB

[3]Cross Entropy: https://en.wikipedia.org/wiki/Secure_Shell

Programming Frameworks

The end-to-end system implementation is done using the python programming language. All the programming tools, programming language, libraries, databases, and frameworks used in this thesis are open source. Hence they can be downloaded and used according to the solution requirement for free. The details of these components are listed below,

- **Programming Language**: Python (V 3.8.10)
- **Programming Tools**: PyCharm community edition (V 2022.1.1), Jupyter Notebook (V 6.4.5), Google Colab
- **Libraries**: Transformers (V 4.19.2), Scikit-learn (V 1.1.0), Numpy (V 1.22.3), Flask (V 2.1.2), NLPAUG (V 1.1.10), PyMongo (4.1.1), Kafka-Python (V 2.0.2)
- **Database**: MongoDB (V 5.0.7)
- **DL Framework**: PyTorch (V 1.11.0), Pandas (V 1.4.2)

5.6 Evaluation Results

The evaluation findings are covered in this section. Conclusions about hypotheses are based on the findings. We first discuss about the evaluation findings for the non-iterative model. The outcomes for each hypothesis are then discussed. In our last section, we talk about the additional findings in the context of the accuracy of our model. We use the dependent variable to evaluate how well our model is working. Each iteration in all the results consists of two cycles. When the model learns the data flawlessly, the train and test losses are not shown in the visualisation of experiment results. However, experiments with poor model training and test losses are shown.

5.6.1 Single Training Batch

Our model is assessed in this section using a single training batch. With regard to our model, the outcomes of this experiment are referred to as SOTA findings. We compare the performance of our iterative model to the batch-trained non-iterative model.

Experiment Setup

This experiment makes use of the movie dataset. As a training batch, we are using 80000 data points. Epochs are set to 10, and the batch size is set to 256. The experiment is carried out for various data skewness (5%, 10%, 20%, and 50%). The other independent variables, such as sample size and sampling strategies are not taken into account since we are training our model in batch mode. The match classes F1 score is used to assess the model.

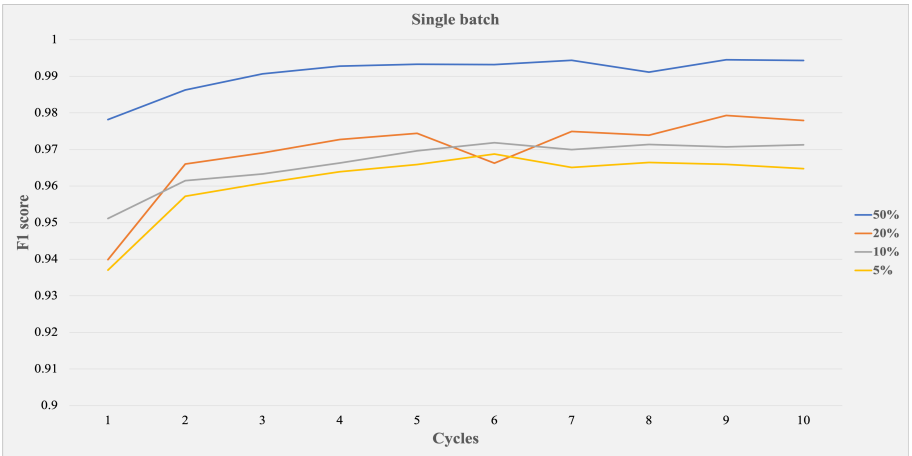


Figure 5.1: Model's performance for varying skewness of single batch.

Results

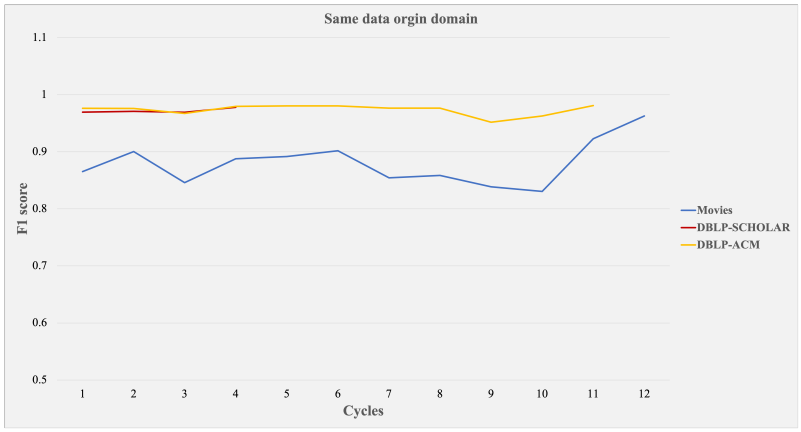
Figure 5.1 shows the epochs on the x-axis and the f1 score of the match class in relation to data skewness on the y-axis. The Figure demonstrates that given fully balanced data, or data with a normal distribution, the greatest f1 score obtained is approximately 0.99. However, the f1 score declines as the data skewness for the not-match class increases. Hence, the model's behaviour is referred to as bias. The model tends to bias toward learning the properties of entity descriptions of not-match data samples. Our model achieves around a 0.96 f1 score even with input data that contains 5%, match classes. The model produces approximately 0.97 and 0.98 f1 scores for the datasets with 10% and 20% entity pairs of match class, respectively. Therefore, we evaluate how well our iterative model performed in relation to these scores.

5.6.2 Hypothesis One

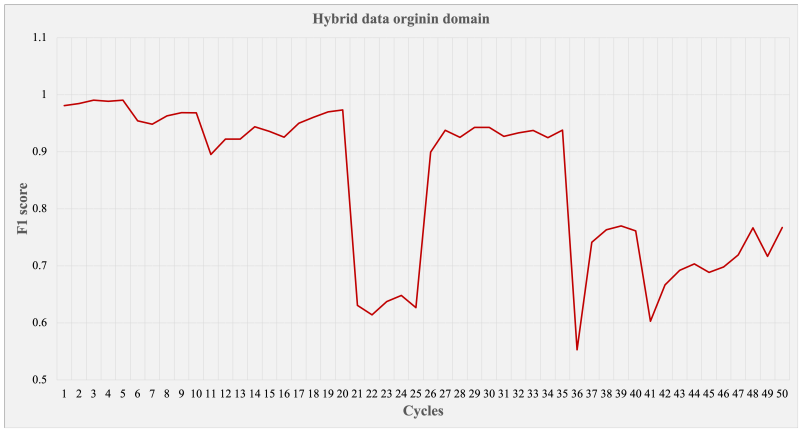
To support the first hypothesis, we evaluate our module in this part, i.e., **"The model quality improves over the iterations if the input data is from the same domain"**.

Experiment Setup

We use conditions C2 and C9 as well as all datasets shown in Table 5.2 to perform this experiment. Movies, DBLP-ACM, and DBLP-SCHOLAR datasets belong to the same domain, while DBPedia datasets belong to the hybrid domain. While using the same independent variables, such as sample size, data skewness, and sampling techniques, conditions C2 and C3 differ in the input data domain. We adopt two different values for the time_window_min parameter along with these independent variables. Since the datasets DBLP-ACM, DBLP-Scholar, and Movies have a small number of candidate pairs, the parameter time_window_min is set to five minutes. To assess the model, we utilize the match class's f1 score. Compared to other dataset streams, we take into account large dataset streams for the DBPedia dataset.



(a)



(b)

Figure 5.2: (a) Model's performance for same domain input. (b) Model's performance for hybrid domain input.

Results

The x-axis in both Figures represents cycles, and the y-axis represents the f1 score of the match class. The f1 scores of the DBLP-Scholars and DBLP-ACM datasets are better than those of the Movies dataset, as seen in Figure 5.2a. The mode's f1 score initially ranges between 0.85 and 0.9, but after cycle ten, the value increases. For the movie dataset, the iterative model gets the maximum f1 score of 0.96, which is comparable to the f1 scores of the non-iterative model (0.96 and 0.97).

The f1-score of the match class for DBPedia datasets is shown in Figure 5.2b. This dataset includes different domains. The model gets f1 score of 0.98 during the first cycles. However, after cycle twenty, when the model came across a dataset from a new domain, the f1 score of the model varies.

Analysis

Figure 5.2a shows that the f1 scores for the DBLP-Scholar and DBLP-ACM datasets have remained not constant but comparably stable over the cycles. For both datasets, the model has the highest f1 score of 0.96. These datasets are made up of citation-representing entities. Both data sources' entities share attribute names (feature) such as title, author, or publication year. The model retrieves these common features efficiently. The attribute names (features) in the Movie dataset vary depending on the data source. For instance, starring is used in the IMDB dataset, while actor name is used in DBPedia as the attribute name. Along with attribute names, each entity has a different amount of attribute names and value pairs. As a result, the model learning task is complicated. The movie dataset's f1 score varies initially in Figure 5.2a but gradually rises to 0.96 over cycles.

The model underperforms on the DBPedia dataset across cycles, as seen in Figure 5.2b. Entity pairs from various domains, including movies, citations, and products, make up the DBPedia. Consequently, the attribute name and values vary for each data point (features). This makes model learning challenging. Another reason for model's poor performance might be the entity descriptions having a series of non-natural language tokens. To examine this hypothesis, we considered the entity descriptions of DBPedia dataset. These entity descriptions consists of URLs[4], which are not a series of natural language tokens.

This study leads us to the conclusion that **hypothesis one is true**, i.e., our model performs well for datasets from the same domain but poorly for datasets from the hybrid domain.

5.6.3 Hypothesis Two

In this section, we evaluate our model to conclude the second hypothesis, i.e., **"The model quality improves over time and takes a smaller number of iterations to converge to near-optimum performance if the upcoming data set is balanced with respect to matches and not-matches"**.

Experiment Setup

This analysis is carried out using the Movies dataset. To transfer the entity tuples from primary collection to secondary collection in CP, the parameter time_window_min is set to five minutes. We take into account conditions C1, C4, and C5, since the model performance is assessed for different data skewness. The strategy and sample size are constant parameters. In these conditions, the training dataset's 5% match entity pair correlates to the high skewness. Low skewness in the training dataset corresponds to 50% match entity pairs, whereas medium skewness is made up of 10% and 20% match entity pairs. The f1 score of match classes is used to assess the model.

[4]Cross Entropy: https://en.wikipedia.org/wiki/URL

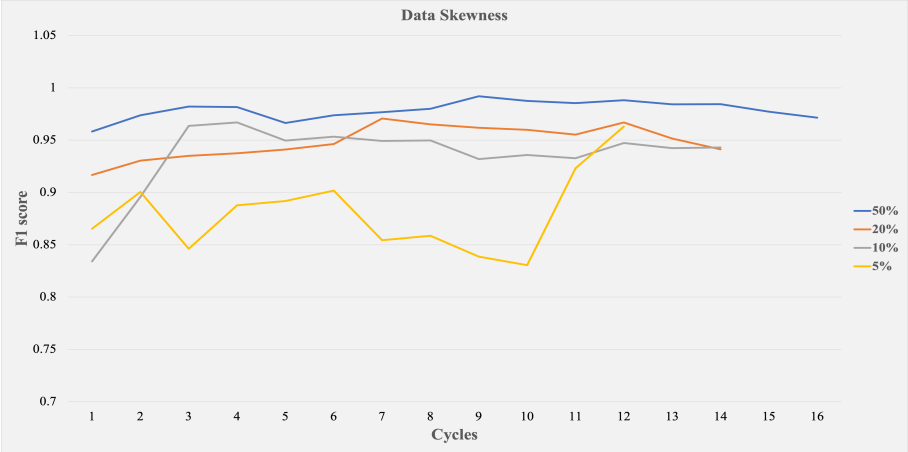


Figure 5.3: Model's performance for varying training data skewness.

Results

The f1 score for the match class of the model across iterations is shown in Figure 5.3. The varying data skewness is also represented in the Figure. The percentage of match entity pairs taken into account in the training sample is shown in the graph. The model works well with training samples that are less skewed (50%) and medium skewed (10% and 20%). The model initially performs poorly for training datasets with 10% match entity pairs, but finally, in the fifth cycle, it achieves a close to optimal f1 score. The model achieves the maximum f1 score of 0.97 for this dataset. Similar to this, with training samples of 50% and 20%, the model achieves a close to optimal f1 score in the third and seventh cycles, respectively. These datasets' highest f1 scores are 0.99 and 0.98, respectively. The model's performance is low for highly skewed data. The model's f1 score ranges between 0.85 and 0.9 in the early cycles for highly skewed data. However, the model starts to perform better after the tenth cycle, achieving the maximum f1 score of 0.96. When compared to the non-iterative model, our iterative model performs quite well. In terms of skewness, iterative models have the same f1 score as non-iterative models.

Analysis

The iterative model performs at par with the non-iterative model. The data that is less skewed has an equal number of match and non-match classifications. Both classes' characteristics are equally presented in the model. However, the model learns the characteristics of a highly skewed class as the skewness of the training data towards one class increases, which leads to significant bias in the model. In the field of machine learning research, this is a well-known issue.

Considering the results shown in Figure 5.3, **hypothesis two holds**, i.e., the model performs better for less skewed training samples. However, the model reaches the greatest f1 score for the 90% skewed dataset (10% match entities) before it does for the 80% skewed dataset (20% match entities). However, compared to prior iterations, the model improves for data which is 80% skewed.

5.6.4 Hypothesis Three

The third hypothesis is derived by assessing our model in this section, i.e., **"The model quality improves over iterations for the large training dataset"**.

Environment Setup

We use the Movie dataset to conduct this experiment. This hypothesis is tested using the conditions C1, C2, and C3. By keeping the other variables constant in this experiment, the training sample size variable is altered. The small, medium, and big numbers, respectively, represent 5000, 10000, and 25000 data points. The primary collection of CP must have more than 25000 predicted data points, which must be transferred to the secondary collection, in order to select the large training sample. Therefore, the parameter time_window_min is set to ten minutes. The model is evaluated both with and without augmentation for a large training sample size. The f1 score variation over iterations is used to assess the model.

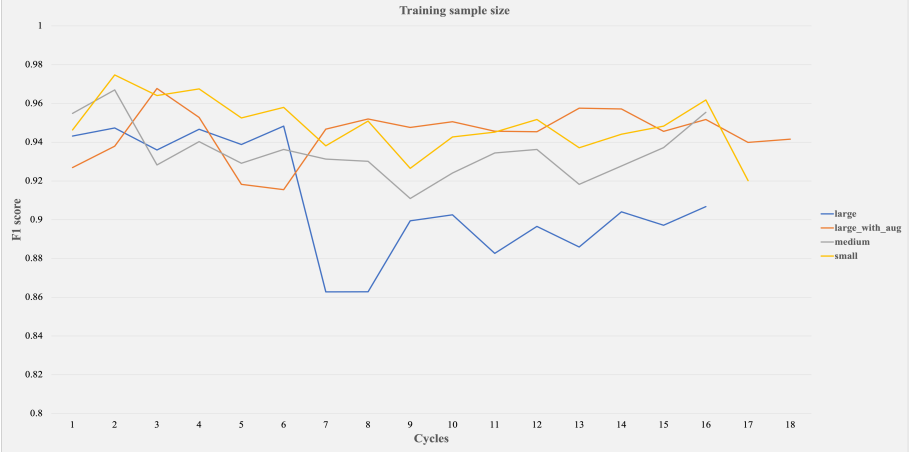


Figure 5.4: Model's performance for varying training sample size.

Results

The f1-score of the match class for various training sample sizes is shown on the y-axis in Figure 5.4 along with the cycles on the x-axis. The use of augmentation for a large training sample is indicated by the label large_with_aug. The model initially outperforms others for the small training sample size, but throughout the iterations, the performance falls short of the augmented large sample size. In the second cycle, the model for this sample size obtains the maximum f1 score, while the f1 score declines to its lowest in the final cycle. Similar results are obtained by the model for the medium sample size. But starting with cycle thirteen, the model performs better for medium sample size, averaging around a 0.96 f1 score. For a large sample size, the augmentation improves the model. The analysis portion provides a detailed explanation for this behaviour. In the sixth cycle, the model's performance for a large sample size reduces to 0.86 f1 scores. The model works better and its performance for enhanced large sample size has increased during the same cycle.

Analysis

The Figure demonstrates how poorly the model performs in the sixth cycle for a large dataset. The performance of the model is thought to be caused by the skewness of the data. For this experiment, the medium skewed data (10% match class) is taken into account. In the sixth iteration, the training sample may have received extremely skewed data, meaning that fewer than 10% of the sample's data points belong to the match target class. This causes the model to learn the features of the not-match class. To study this behaviour, the augmentation strategy is used on a large sample set. After utilizing the augmentation, the model's performance is shown in the Figure. An enhanced large sample improves the model's performance. This validated our assumption.

We **cannot draw a conclusion** for hypothesis three since the model's performance for enhanced large and small sample sets is roughly comparable across iterations. Using the augmentation strategy is one way to enhance the model for a large sample set. This topic will be the subject of further study.

5.6.5 Hypothesis Four

In this, we discuss the evaluation of our model to conclude hypothesis four, i.e., **"The model quality varies over iterations for multiple sampling strategies that fetch the data set for labelling"**.

Experiment Setup

The model is assessed using the Movie dataset. The parameter time_window_min is set to five minutes. Multiple sampling techniques are used in this experiment to subsample the dataset for labelling. The data set used to train the model is then annotated by the oracle. The conditions C2, C6, C7, and C8, are used to assess the model in the thesis, where we offer four sampling strategies that are described in depth in Section 4.3.2. Strategies three, zero, one, and two are taken into consideration by the conditions C2, C6, C7, and C8, respectively. Other factors like sample size and data skewness are set to medium and 90%, respectively. The f1 score was used to assess the model.

Results

The model's performance for various strategies is shown in Figure 5.5. In Section 4.3.4, we mentioned that the model is not trained if the trained model from the previous iteration outperforms it (the f1 score and test loss are greater than those of the preceding model, or the predetermined threshold). As a result, from cycle four to cycle thirteen for strategy1, the model's f1 score is absent in Figure 5.5. The model initially performs better for strategy0, achieving a roughly 0.97 f1 score. However, the model's performance declines to a 0.93 f1 score in the last two rounds. The models' performance improves across the cycles for the data sample obtained by using strategy0. In the ninth cycle, the model gets a f1 score of 0.97. For the training sample retrieved using strategy3, the model exhibits the same behaviour. However, in the eleventh cycle, the model's performance decreases to a 0.96 f1 score. For the training sample of strategy2, the model performs badly. The model varies between iterations, but it reaches the greatest f1 score of 0.96.

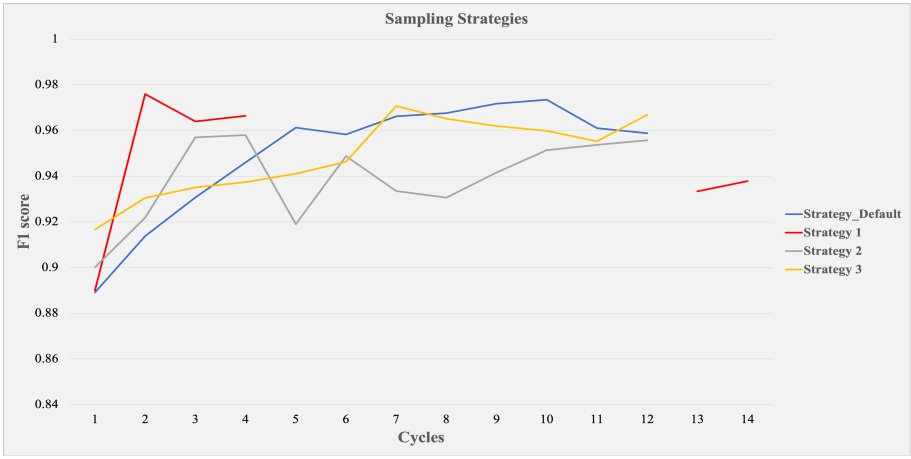


Figure 5.5: Model's performance for different sampling strategies.

Analysis

In this experiment, we take into account a number of strategies that retrieves sample dataset from the candidate pool for annotation using various methodologies. In Subsection 4.3.2, specific explanations of the techniques are provided. Below, we analyze the outcomes from the perspective of the chosen strategy.

1. **Strategy0**: There is a good likelihood that samples will have a larger proportion of entity pairs from the match class since the candidate pool's data points are ordered according to similarity score (w_{ab}). The model performs better for a more balanced training sample, as we have shown in hypothesis two. As a result, the model works better with this tactic.

2. **Strategy1**: The first N entity tuples with the greatest confidence are included in the sample. The likelihood of categorizing an entity pair is determined by the model confidence. The model's confidence in classification increases with increasing likelihood. The similarity score determined by the cosine similarity function is the confidence value given to each entity tuple in the first two cycles. The cosine similarity function's efficiency in model training is shown in the first two cycles. The confidence is assigned based on the probability determined from the model after it has been trained. The chance of containing entity tuples of the match class is lower because the model classifies both classes with better confidence. As a result, the model has performed badly during the last two cycles.

3. **Strategy2**: By choosing the entity pairs with the greatest and lowest model confidence, this technique aims to solve the issue of reduced probability of picking match class's entity pairs. The results demonstrate that the model underperforms for strategy2, contradicting its purpose.

4. **Strategy3**: By randomly generating three bins and choosing a certain number of entity pairs from each bin, the technique addresses the issue of the match class entity pairs being less likely to be chosen. The outcome demonstrates that the tactic solves the issue. The training sample is more evenly distributed with this method, and the model's performance increases over iterations.

5.6.6 Additional Findings

In this section, we go through the additional experiments that are used to assess the model performance for sequential input data and data augmentation. Additionally, each part of our model's execution time is described.

1. **Sequential-domain input data**
   In this experiment, we aim to study the model's performance for the input data representing multiple domains sequentially.

   Experiment Setup: The iterative training model is evaluated in this part using input from the first *N* entity pairs from the citation (DBLP-ACM) domain and the subsequent entity pairs from the Movies domain. The data skewness is set to medium (10%), the training sample size is set to small, the sampling strategy is set to the third strategy, and the time_window_min is set to five minutes. We are not employing the augmentation approach since we are using data that is moderately skewed. The f1 score, test loss, and train loss are used to assess the model performance.

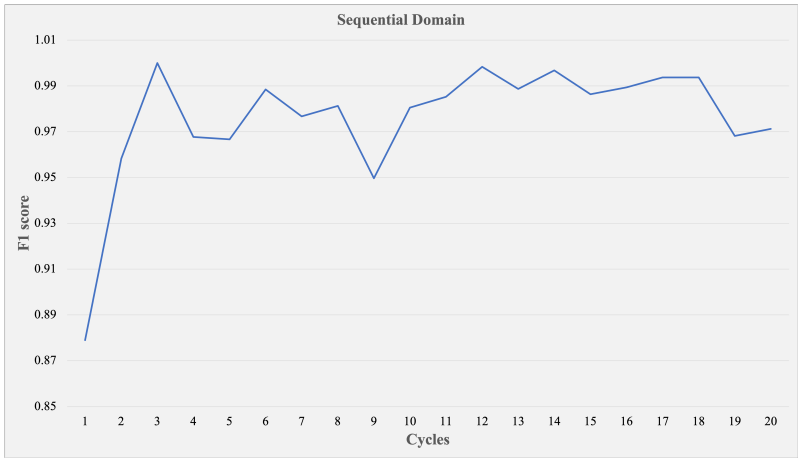
   Results: The model's f1-score of match classes is shown in Figure 5.6a. The iterative model initially performs better. In the third cycle, the model receives the highest f1 score of 1, but after that, its performance falls by 0.04. The model performs poorly in the ninth cycle, dropping the f1 score to 0.95. However, the model's performance improves and achieves an approximately 1 f1 score. Performance of the model varies between 0.95 and 1.

   The training loss and test loss of the model are shown in Figure 5.6b. The Figure shows that the model's training loss decreases till cycle eight and then increases. The loss reaches 0.14 (14%) for each cycle. The model's test loss, however, does not vary as much as its training loss. In the seventh cycle, the model's test loss increases to 0.05.

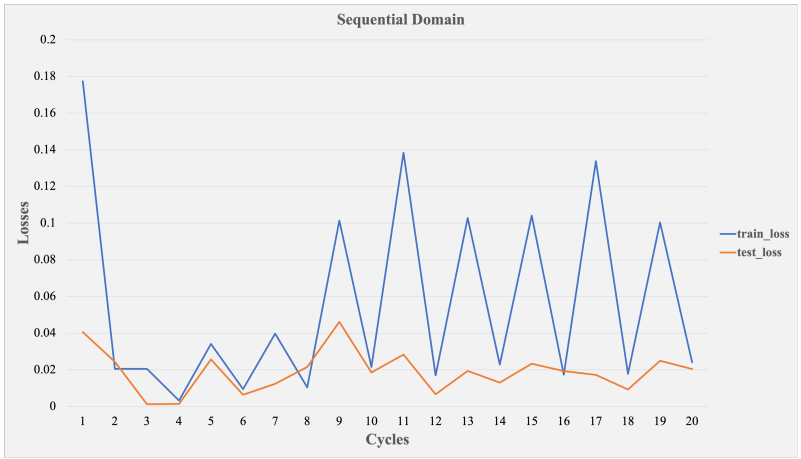
   Analysis: The model performs poorly in the eighth cycle across all the Figures. The entity tuples from the citation domain are the first entities the model runs into. As a result, the model learns from the citation domain's characteristics. When the model comes across entity tuples from the movies domain after the ninth cycle, it attempts to incorporate the new features. As a result, the model exhibits poor performance in the eighth cycle. Figure 5.6a shows that the model performs better at transferring information from one domain to another while learning it. However, we can see in Figure 5.6b that the training error is greater than the test error. The potential causes of this behaviour might be (1) a large number of complex characteristics in the training set that need to be learned and (2) non-complex features in the test set that need to be predicted. Future work would be an in-depth investigation of this.

2. **Augmentation of the training dataset**
   In this experiment, we evaluate our model's performance for varying augmented entity pairs in training batch.

   Experiment Setup: In this experiment, the percentage of augmented entity pairs in the training sample is varied while using the movie dataset. The input stream consists of 2.5% entity pairs belonging to match class. The parameter time_window_min is set to ten minutes. The independent variables, sample size and sampling strategies are set to medium and third



(a)



(b)

Figure 5.6: (a) Model's performance for sequential input domain. (b) Model's training and test loss for sequential input domain.

strategy, respectively. The 5% augmentation value indicates the percentage of entity pairs in training samples of the match class after applying augmentation approaches. The 10% augmentation is used in a similar way. The f1 score is used to assess the model performance.

Results: The impact of augmentation on the model's performance is seen in Figure 5.7. The model performs between 0.85 and 0.9 f1 scores for the training dataset without augmentation. In the ninth cycle, the model receives its maximum f1 score of 0.91. In contrast, in the eleventh cycle, the performance declines to its lowest f1 score of 0.85. For the 10% of augmented entity pairs, the model performs poorly. The model initially earns the greatest f1 score of 1, but after the fifth cycle, it starts to perform poorly. For 5% of augmented entity pairs, the model performs better. The model performs well in the first few cycles before deteriorating in the middle cycles and reaching a ninth cycle performance f1 score of 0.87. The model's f1 score improves to 0.93 after the ninth cycle, which is better than the model's performance for the training data set without augmentation.

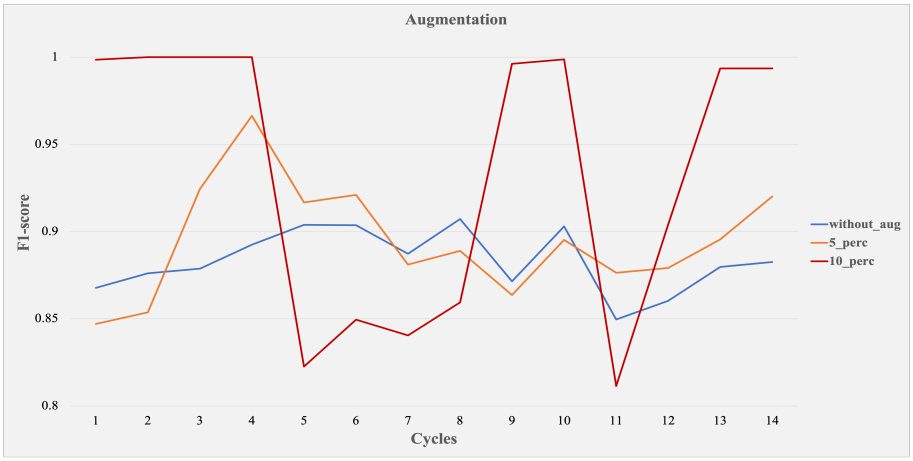


Figure 5.7: Model's performance for different percentage of augmented entity pairs in training sample.

Analysis: Figure 5.7's representation of the model performance demonstrates how the augmentation approach for training data enhances the model's performance up to a saturation percentage (5%). The model, therefore, exhibits poor performance for a greater proportion of augmented entity pairings (10%). We use a duplicate entity technique, which is an augmentation technique covered in Section 4.3.3. The entity pairs utilize identical attribute names and values as a result of this strategy. Then, these entity pairings are subjected to the augmentation approaches. The model learns the features that have the same set of attribute names and values in entity pairs as the number of augmented entity pairs increases. As a result, the model favours these features. The model functions poorly when classifying the match class's continuous streaming entity pairs.

3. **Execution time of components**
   This experiment calculates how long it takes the system and each component to complete an operation. So, it takes care of the second goal of evaluation. We adopt the experimental setup used in to test hypothesis three. Each component's average execution time across the iterations is taken into account. The unit of measuring the execution time is in "seconds".

   The average execution time for each training layer component is shown in Figure 5.8. For various training sample sizes, the execution time for strategy application, model storage, and model deployment remain constant. These parts are connected to the computer system's hardware. However, compared to other components, data labelling requires the most time. The number of entity tuples to annotate grows along with the size of the training sample; as a result, the annotation process takes longer. The size of the training sample has an impact on both model training and augmentation. The system execution time is 123.4, 175.12, and 335.39 seconds for small, medium, and high training sample sets, respectively.

   The trained model is deployed using the Flask API to classify the streaming entity pairs, as it was covered in Subsection 4.3.5. An entity pair is predicted by the model in 0.013 seconds. The prediction layer processes 1000 records in 12.5 seconds at the same time. This duration is determined by timing the transfer of 1000 records from the lookup table to the candidate pool.

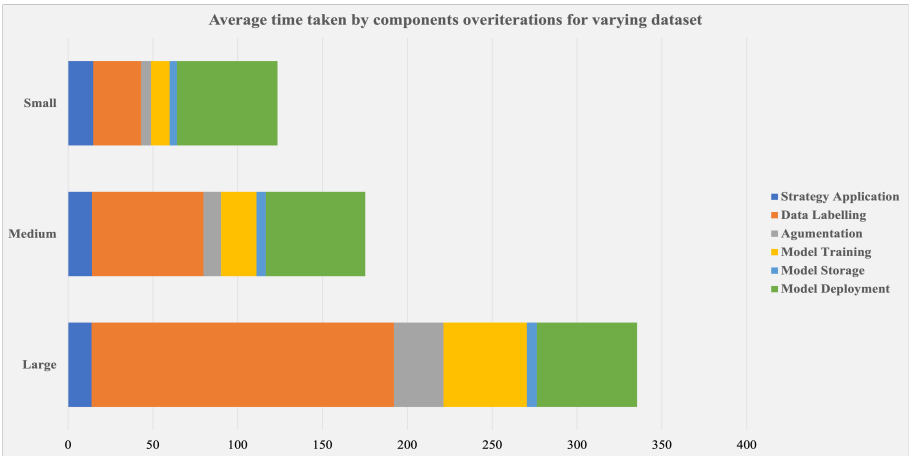


Figure 5.8: Average execution time taken by components.

6 Conclusion and Future Work

Through the course of this thesis, we mainly studied the incremental ER method that is iteratively trained on the newer dataset and classifies the continuously streaming entity pairs. Different aspects of classification function, active learning to annotate the data points, augmentation to resolve the data imbalance problem, and iterative model training were explored and implemented. In order to decouple the classification of the continuous streaming entity pairs and train the model iteratively, a prediction layer and a training layer are designed and implemented. The system was assessed in an evaluation, and the experiment results are presented.

In order to summarise the work, research objectives are revisited.

1. **To address the problem of entity resolution for streaming entity pairs.**
   A prediction layer for continuous streaming entity pair classification was developed and implemented. We discussed an effective method of retrieving entity descriptions using look-up tables as the first component of this layer. The serialization technique, which converts the input entity description's attribute name and value pairs into a sequence of tokens, is then discussed. The classification function, a crucial component of the prediction layer, is finally described. The prediction layer classifies the continuous streaming entity pairs as match or not match using all of these elements.

2. **To explore the concept of continuous iterative training to achieve ER in the case of continuous streaming entity pairs.**
   In connection with a previous study, it was discovered that the transformer-based pre-trained language model is the most recent and effective way of classifying continuous streaming entity pairs. It helps in addressing the entity pair heterogeneity issue in continuous streaming. The features of the most recent input entity pairs are trained to a pre-trained language model called DistilBERT through iterative learning. In order to convert the text input into machine-readable vectors, the model leverages previously learned information. The same vectors are subsequently used to train the task-specific layer. This model is better and does not need all of the training data at once since it is trained iteratively.

3. **To explore the role of active learning and crowd-sourcing-based ER in incremental model training.**
   Key components of incremental learning are active learning and crowdsourcing-based ER techniques because they provide accurate labels for entity pair labels for model learning. In order to subsample the entity pairs from the population for annotation of entity pairs, we developed four strategies. As a part of the implementation of the data labelling component, we discussed the human Oracle-based approach for providing the true labels. Due to time restrictions, we were unable to use a real human to annotate the data in our thesis. Instead, we created a script that annotated entity pairs using the ground truth files. These annotated data sets are offered as training sets for building models.

4. **To evaluate the end-to-end streaming entity resolution system developed in this thesis and analyse its performance**
   The evaluation of the model's performance involved varying the independent variables and observing the change in dependent variables. We introduced nine conditions to evaluate the model's performance by varying the input data with respect to domain, size, skewness, and sampling strategies. The four hypotheses were discussed to analyse the model's performance. We conducted several experiments to conclude these hypotheses. From these experiments, we observed that our iteratively trained model achieves a performance similar to the model that is trained once with all the training samples.

In conclusion, this thesis demonstrates that the transformer-based pre-trained language models can be trained iteratively to learn the features of newer data and classify the continuously streaming entity pairs. Evaluation findings shed light on the performance of the streaming entity resolution system and the role of different components in reference resolution. The results and challenges faced during the implementation hints to us at how to make the system better and scope for future work.

Future Work

The methods used in this thesis for model deployment, augmentation, and classification function all have certain drawbacks. They are covered in the Chapter 4. This thesis's design is extremely adaptable, and other components may be included with ease. We now discuss several possible directions for future research.

1. The transformer-based pre-trained language model in our design leverages DistilBERT to encode entity pairs. Other variations of BERT, such as BERT and RoBERTa, might take the place of this model in the next studies. The research may compare the related results.
2. Similar to the above, different similarity functions such as the Jaccard function can be applied to enhance the execution time of the prediction layer.
3. In our work, augmentation is applied if the percentage of entity pairs of match class is less than the threshold (Subsection 4.3.3). It involves the augmentation application if the data set does not consist of the match class's entity pairs. In future work, this limitation can be addressed by discarding such batches.
4. As explained in Subsection 4.3.3, we use a duplicate entity to oversample the entity pairs of match class. Hence, the model is biased towards the same features in both entities. This leads to the model's poor performance in classifying the continuously streaming entity pairs. In the same section, we discussed different augmentation techniques such as key-board errors, spelling mistakes, or synonym replacement. NLPAUG library consists of other augmentation techniques such as contextual word embedding augmentation (BERT, DistilBERT, or RoBERTa), TF-IDF augmentation, or word embedding augmentation (word2vec, GloVe, or FastText). The techniques could be addressed in future work.
5. The model's performance is measured during the evaluation using the nine conditions drafted from the evaluation metrics. There are more possibilities for the different combinations of these variables. These combinations could be addressed in future work.

Bibliography

[ALM17] S. Arora, Y. Liang, T. Ma. "A simple but tough-to-beat baseline for sentence embeddings". In: *International conference on learning representations*. 2017 (cit. on p. 27).

[BBS05] M. Bilenko, S. Basil, M. Sahami. "Adaptive product normalization: Using online learning for record linkage in comparison shopping". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE. 2005, 8–pp (cit. on p. 19).

[BCB14] D. Bahdanau, K. Cho, Y. Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 25).

[BDNW12] C. Böhm, G. De Melo, F. Naumann, G. Weikum. "LINDA: distributed web-of-data-scale entity matching". In: *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2012, pp. 2104–2108 (cit. on p. 21).

[BGJM17] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov. "Enriching word vectors with subword information". In: *Transactions of the association for computational linguistics* 5 (2017), pp. 135–146 (cit. on pp. 25, 26).

[BMR11] P. A. Bernstein, J. Madhavan, E. Rahm. "Generic schema matching, ten years later". In: *Proceedings of the VLDB Endowment* 4.11 (2011), pp. 695–701 (cit. on p. 20).

[CEP+19] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, K. Stefanidis. "End-to-end entity resolution for big data: A survey". In: *arXiv preprint arXiv:1905.06397* (2019) (cit. on pp. 16, 19).

[CEP+20] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, K. Stefanidis. "An overview of end-to-end entity resolution for big data". In: *ACM Computing Surveys (CSUR)* 53.6 (2020), pp. 1–42 (cit. on pp. 15, 20, 22, 31).

[CES15] V. Christophides, V. Efthymiou, K. Stefanidis. "Entity resolution in the web of data". In: *Synthesis Lectures on the Semantic Web* 5.3 (2015), pp. 1–122 (cit. on pp. 19–21).

[Che15] Y. Chen. "Convolutional neural network for sentence classification". MA thesis. University of Waterloo, 2015 (cit. on p. 26).

[Chr] P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. 2012* (cit. on p. 19).

[Chr12] P. Christen. "The data matching process". In: *Data matching*. Springer, 2012, pp. 23–35 (cit. on p. 20).

[CKLM19] K. Clark, U. Khandelwal, O. Levy, C. D. Manning. "What does bert look at? an analysis of bert's attention". In: *arXiv preprint arXiv:1906.04341* (2019) (cit. on p. 29).

[CKLS01] M. Cochinwala, V. Kurien, G. Lalk, D. Shasha. "Efficient data reconciliation". In: *Information Sciences* 137.1-4 (2001), pp. 1–15 (cit. on p. 16).

[DCLT18] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on pp. 17, 29, 43, 53).

[DG08] J. Dean, S. Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on p. 21).

[DS15] X. L. Dong, D. Srivastava. “Big data integration”. In: *Synthesis Lectures on Data Management* 7.1 (2015), pp. 1–198 (cit. on pp. 16, 21).

[EIV06] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios. “Duplicate record detection: A survey”. In: *IEEE Transactions on knowledge and data engineering* 19.1 (2006), pp. 1–16 (cit. on p. 20).

[EKR+21] Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, Y. Goldberg. “Measuring and improving consistency in pretrained language models”. In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 1012–1031 (cit. on p. 28).

[EPSC19] V. Efthymiou, G. Papadakis, K. Stefanidis, V. Christophides. “MinoanER: Schema-agnostic, non-iterative, massively parallel resolution of web entities”. In: *arXiv preprint arXiv:1905.06170* (2019) (cit. on pp. 23, 33, 35).

[ETJ+18] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, N. Tang. “Distributed representations of tuples for entity resolution”. In: *Proceedings of the VLDB Endowment* 11.11 (2018), pp. 1454–1467 (cit. on pp. 19, 28, 33, 35).

[FS69] I. P. Fellegi, A. B. Sunter. “A theory for record linkage”. In: *Journal of the American Statistical Association* 64.328 (1969), pp. 1183–1210 (cit. on pp. 19, 20, 24).

[GDS14] A. Gruenheid, X. L. Dong, D. Srivastava. “Incremental record linkage”. In: *Proceedings of the VLDB Endowment* 7.9 (2014), pp. 697–708 (cit. on p. 21).

[GH20] L. Gazzarri, M. Herschel. “Towards task-based parallelization for entity resolution”. In: *SICS Software-Intensive Cyber-Physical Systems* 35.1 (2020), pp. 31–38 (cit. on p. 35).

[GH21] L. Gazzarri, M. Herschel. “End-to-end Task Based Parallelization for Entity Resolution on Dynamic Data”. In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021, pp. 1248–1259 (cit. on pp. 15, 16, 21, 31, 34, 37).

[GHHY96] O. Goldschmidt, D. S. Hochbaum, C. Hurkens, G. Yu. “Approximation algorithms for the k-clique covering problem”. In: *SIAM Journal on Discrete Mathematics* 9.3 (1996), pp. 492–509 (cit. on p. 33).

[Gil01] L. Gill. *Methods for automatic record matching and linkage and their use in national statistics*. 25. Office for National Statistics, 2001 (cit. on p. 19).

[GM12] L. Getoor, A. Machanavajjhala. “Entity resolution: theory, practice & open challenges”. In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 2018–2019 (cit. on p. 16).

[HS97] S. Hochreiter, J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 28).

[INN08] E. Ioannou, C. Niederée, W. Nejdl. “Probabilistic entity linkage for heterogeneous information spaces”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2008, pp. 556–570 (cit. on pp. 24, 25, 33, 35).

[JN07] F. V. Jensen, T. D. Nielsen. *Bayesian networks and decision graphs*. Vol. 2. Springer, 2007 (cit. on p. 25).

[Joa96] T. Joachims. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* Tech. rep. Carnegie-mellon univ pittsburgh pa dept of computer science, 1996 (cit. on p. 44).

[KB14] D. P. Kingma, J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 54).

[KDD+16] P. Konda, S. Das, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, et al. "Magellan: toward building entity matching management systems over data science stacks". In: *Proceedings of the VLDB Endowment* 9.13 (2016), pp. 1581–1584 (cit. on p. 26).

[KR10] H. Köpcke, E. Rahm. "Frameworks for entity matching: A comparison". In: *Data & Knowledge Engineering* 69.2 (2010), pp. 197–210 (cit. on p. 19).

[KTR10] H. Köpcke, A. Thor, E. Rahm. "Evaluation of entity resolution approaches on real-world match problems". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 484–493 (cit. on p. 20).

[KTR12] L. Kolb, A. Thor, E. Rahm. "Dedoop: Efficient deduplication with hadoop". In: *Proceedings of the VLDB Endowment* 5.12 (2012), pp. 1878–1881 (cit. on p. 21).

[LBH+15] Y. LeCun, Y. Bengio, G. Hinton, et al. "Deep learning. nature, 521 (7553), 436-444". In: *Google Scholar Google Scholar Cross Ref Cross Ref* (2015) (cit. on p. 25).

[LH17] I. Loshchilov, F. Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017) (cit. on p. 54).

[LLG14] L. Li, J. Li, H. Gao. "Rule-based method for entity resolution". In: *IEEE Transactions on Knowledge and Data Engineering* 27.1 (2014), pp. 250–263 (cit. on p. 16).

[LLS+20] Y. Li, J. Li, Y. Suhara, A. Doan, W.-C. Tan. "Deep entity matching with pre-trained language models". In: *arXiv preprint arXiv:2004.00584* (2020) (cit. on pp. 16, 30, 33, 35, 37, 39, 43, 55, 63).

[LPD+13] S. Lacoste-Julien, K. Palla, A. Davies, G. Kasneci, T. Graepel, Z. Ghahramani. "Sigma: Simple greedy matching for aligning large knowledge bases". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 572–580 (cit. on p. 21).

[Ma19] E. Ma. *NLP Augmentation*. https://github.com/makcedward/nlpaug. 2019 (cit. on pp. 39, 52).

[MBR01] J. Madhavan, P. A. Bernstein, E. Rahm. "Generic schema matching with cupid". In: *vldb*. Vol. 1. Citeseer. 2001, pp. 49–58 (cit. on p. 20).

[MLR+18] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra. "Deep learning for entity matching: A design space exploration". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 19–34 (cit. on pp. 16, 26, 27, 33, 35).

[Moh20] M. Mohammadkhani. *A Comparative Evaluation of Deep Learning based Transformers for Entity Resolution*. Tech. rep. 2020 (cit. on pp. 16, 19).

[MS04] A. McCallum, C. Sutton. “Piecewise training with parameter independence diagrams: Comparing globally-and locally-trained linear-chain crfs”. In: *NIPS 2004 Workshop on Learning with Structured Outputs*. 2004 (cit. on p. 16).

[MSC+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems* 26 (2013) (cit. on p. 26).

[NHH+19] H. Nie, X. Han, B. He, L. Sun, B. Chen, W. Zhang, S. Wu, H. Kong. “Deep sequence-to-sequence entity matching for heterogeneous entity resolution”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 629–638 (cit. on pp. 25, 26, 33, 35).

[OR18] M. Odell, R. Russell. “The soundex coding system”. In: *US Patents* 1261167 (1918), p. 9 (cit. on p. 24).

[PHN14] T. Papenbrock, A. Heise, F. Naumann. “Progressive duplicate detection”. In: *IEEE Transactions on knowledge and data engineering* 27.5 (2014), pp. 1316–1329 (cit. on p. 21).

[PIP+12] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, W. Nejdl. “A blocking framework for entity resolution in highly heterogeneous information spaces”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.12 (2012), pp. 2665–2682 (cit. on p. 21).

[PIP20] G. Papadakis, E. Ioannou, T. Palpanas. “Entity Resolution: Past, Present and Yet-to-Come.” In: *EDBT*. 2020, pp. 647–650 (cit. on pp. 20–22, 33).

[PN11] G. Papadakis, W. Nejdl. “Efficient entity resolution methods for heterogeneous information spaces”. In: *2011 IEEE 27th International Conference on Data Engineering Workshops*. IEEE. 2011, pp. 304–307 (cit. on p. 21).

[PSM14] J. Pennington, R. Socher, C. D. Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543 (cit. on p. 26).

[PTDU16] A. P. Parikh, O. Täckström, D. Das, J. Uszkoreit. “A decomposable attention model for natural language inference”. In: *arXiv preprint arXiv:1606.01933* (2016) (cit. on p. 27).

[RPHP17] O. F. Reyes-Galaviz, W. Pedrycz, Z. He, N. J. Pizzi. “A supervised gradient-based learning algorithm for optimized entity resolution”. In: *Data & Knowledge Engineering* 112 (2017), pp. 106–129 (cit. on pp. 24, 33, 35).

[RWC+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 29).

[RZLL16] P. Rajpurkar, J. Zhang, K. Lopyrev, P. Liang. “Squad: 100,000+ questions for machine comprehension of text”. In: *arXiv preprint arXiv:1606.05250* (2016) (cit. on p. 17).

[SAS11] F. M. Suchanek, S. Abiteboul, P. Senellart. “Paris: Probabilistic alignment of relations, instances, and schema”. In: *arXiv preprint arXiv:1111.7164* (2011) (cit. on p. 21).

[SBJ16] G. Simonini, S. Bergamaschi, H. Jagadish. “BLAST: a loosely schema-aware meta-blocking approach for entity resolution”. In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 1173–1184 (cit. on p. 21).

[SDCW19] V. Sanh, L. Debut, J. Chaumond, T. Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019) (cit. on pp. 17, 37, 54).

[Set09] B. Settles. “Active learning literature survey”. In: (2009) (cit. on p. 49).

[SP97] M. Schuster, K. K. Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681 (cit. on pp. 27, 28).

[SPPB18] G. Simonini, G. Papadakis, T. Palpanas, S. Bergamaschi. “Schema-agnostic progressive entity resolution”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.6 (2018), pp. 1208–1221 (cit. on p. 21).

[SPW+13] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642 (cit. on p. 17).

[Tal11] J. R. Talburt. *Entity resolution and information quality*. Elsevier, 2011 (cit. on p. 19).

[TDP19] I. Tenney, D. Das, E. Pavlick. “BERT rediscovers the classical NLP pipeline”. In: *arXiv preprint arXiv:1905.05950* (2019) (cit. on p. 29).

[TJ13] V. Thada, V. Jaglan. “Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm”. In: *International Journal of Innovations in Engineering and Technology* 2.4 (2013), pp. 202–205 (cit. on p. 45).

[TSS20] K.-S. Teong, L.-K. Soon, T. T. Su. “Schema-Agnostic Entity Matching Using Pre-Trained Language Models”. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2241–2244 (cit. on pp. 29, 33, 35, 39).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 29).

[WKFF12] J. Wang, T. Kraska, M. J. Franklin, J. Feng. “Crowder: Crowdsourcing entity resolution”. In: *arXiv preprint arXiv:1208.1927* (2012) (cit. on pp. 32, 34, 35, 49).

[ZCF+10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica. “Spark: Cluster computing with working sets”. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. 2010 (cit. on p. 23).

[Žil06] A. Žilinskas. *Practical mathematical optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms*. 2006 (cit. on p. 24).

All links were last followed on July 8, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature