CLNS – Cloud and Network Infrastructure, EIT Digital

Master's Degree Project, 30hp

Finding duplicate offers in the online marketplace catalogue using transformer based methods

An exploration of transformer based methods for the task of entity resolution

ROBERT-ANDREI DAMIAN

Stockholm, Sverige

Finding duplicate offers in the online marketplace catalogue using transformer based methods

An exploration of transformer based methods for the task of entity resolution

ROBERT-ANDREI DAMIAN

Date: June 26, 2022

Supervisor: Francisco J. Peña
Examiner: Mihhail Matskin
School of Electrical Engineering and Computer Science
Host company: Panprices AB
Swedish title: Hitta dubbletter av erbjudanden i online marknadsplatskatalog med hjälp av transformer-baserade metoder
Swedish subtitle: En utforskning av transformer-baserad metoder för uppgiften att deduplicera

Abstract

The amount of data available on the web is constantly growing, and e-commerce websites are no exception. Considering the abundance of available information, finding offers for the same product in the catalogue of different retailers represents a challenge. This problem is an interesting one and addresses the needs of multiple actors. A customer is interested in finding the best deal for the product they want to buy. A retailer wants to keep up to date with the competition and adapt its pricing strategy accordingly. Various services already offer the possibility of finding duplicate products in catalogues of e-commerce retailers, but their solutions are based on matching a Global Trade Identification Number (GTIN). This strategy is limited because a GTIN may not be made publicly available by a competitor, may be different for the same product exported by the manufacturer to different markets or may not even exist for low-value products. The field of Entity Resolution (ER), a sub-branch of Natural Language Processing (NLP), focuses on solving the issue of matching duplicate database entries when a deterministic identifier is not available. We investigate various solutions from the the field and present a new model called *Spring R-SupCon* that focuses on low volume datasets. Our work builds upon the recently introduced model, R-SupCon, introducing a new learning scheme that improves R-SupCon's performance by up to 74.47% F1 score, and surpasses Ditto by up 12% F1 score for low volume datasets. Moreover, our experiments show that smaller language models can be used for ER with minimal loss in performance. This has the potential to extend the adoption of Transformer-based solutions to companies and markets where datasets are difficult to create, like it is the case for the Swedish marketplace Fyndiq.

Keywords

Transformers, Language Models, Deep Neural Networks, Entity Resolution, Duplicate Detection, Entity Matching, Record Linkage, Contrastive Learning, e-commerce

Sammanfattning

Mängden data på internet växer konstant och e-handeln är inget undantag. Konsumenter har idag många valmöjligheter varifrån de väljer att göra sina inköp från. Detta gör att det blir svårare och svårare att hitta det bästa erbjudandet. Även för återförsäljare ökar svårigheten att veta vilken konkurrent som har lägst pris. Det finns tillgängliga lösningar på detta problem men de använder produktunika identifierare såsom Global Trade Identification Number (förkortat “GTIN”). Då det finns en rad utmaningar att bara förlita sig på lösningar som baseras på GTIN behövs ett alternativt tillvägagångssätt. GTIN är exempelvis inte en offentlig information och identifieraren kan dessutom vara en annan när samma produkt erbjuds på en annan marknad. Det här projektet undersöker alternativa lösningar som inte är baserade på en deterministisk identifierare. Detta projekt förlitar sig istället på text såsom produktens namn för att fastställa matchningar mellan olika erbjudanden. En rad olika implementeringar baserade på maskininlärning och djupinlärning studeras i detta projekt. Projektet har dock ett särskilt fokus på “Transformer”-baserade språkmodeller såsom BERT. Detta projekt visar hur man generera proprietär data. Projektet föreslår även ett nytt inlärningsschema och bevisar dess fördelar.

Nyckelord

Transformers, Språkmodeller, Djupinlärning, Entitetserkännande, Dubblett-detektering, Entitetsmatchning, Rekordkoppling, e-handel

Résumé

Le volume des données qui se trouve sur l'internet est en une augmentation constante et les commerces électroniques ne font pas note discordante. Le consommateur a aujourd'hui beaucoup des options quand il decide d'où faire son achat. Trouver le meilleur prix devient de plus en plus difficile. Les entreprises qui gerent cettes plates-formes ont aussi la difficulté de savoir en tous moments lesquels de ses concurrents ont le meilleur prix. Il y-a déjà des solutions en ligne qui ont l'objectif de résoudre ce problème, mais ils utilisent un identifiant de produit unique qui s'appelle Global Trade identification number (ou GTIN). Plusieurs difficultés posent des barriers sur cette solution. Par exemple, GTIN n'est pas public peut-être, ou des GTINs différents peut-être assigne par la fabricante au même produit pour distinguer des marchés différents. Ce projet étudie des solutions alternatives qui ne sont pas basées sur avoir un identifiant unique. On discute des methods qui font la décision en fonction du nom des produits, en utilisant des algorithmes d'apprentissage automatique ou d'apprentissage en profondeur. Le projet se concentre sur des solutions avec "Transformer" modèles de langages, comme BERT. On voit aussi comme peut-on créer un ensemble de données propriétaire pour enseigner le modèle. Finalement, une nouvelle method d'apprentissage est proposée et analysée.

Mots-clés

Transformers, Modèles de langage, Apprentisage en profondeur, Résolution d'entité, Détection de doublons, Apprentisage contrastif, commerce électronique

Acknowledgments

I would like to thank my supervisor, Francisco J. Peña, without whose support this project wouldn't have been nearly as well implemented and documented. I would also like to thank Panprice's team for being very open and supporting me throughout the period of the project.

I am deeply grateful to Oracle for having awarded me the Cloud Starter Award, consisting of credits on Oracle Cloud. The experiments carried out as part of this project would have been too expensive for me to support the cost. Their intuitive and easy to use cloud platform not only enabled me to perform my work, but it also guaranteed that I was able to do so at the speed required to finish this project in the limited amount of time allocated.

Special thanks go to Ralph Peeters from the University of Mannheim, whose papers represent the foundation of the practical part of this project. Not only he and his colleagues made their code available on Github for other researchers to quickly pick up and improve upon their work, but he also agreed to take a call with me where we exchanged impressions about the field. Having the ability to have such a conversation with a respected scientist from another university meant a lot to me and boosted my confidence as a Master's student.

Stockholm, June 2022
Robert-Andrei Damian

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

ALBERT	A Lite BERT
AUC	Area Under the Curve
BCE	Binary Cross Entropy
BERT	Bidirectional Encoder Representations from Transformer
BiGRU	Bidirectional Gated Recurrent Unit (GRU)
BiLSTM	Bidirectional Long Short Term Memory (LSTM)
BoW	Bag of Words
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
DNN	Deep Neural Network
DT	Decision Tree
ELMo	Embeddings from Language Models
ER	Entity Resolution
FFNN	Feed Forward Neural Network
FNR	False Negative Rate
FPR	False Positive Rate
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
GPT	Generative Pre-training Transformer
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
GTIN	Global Trade Item Number
KB	Knowledge Base
LM	Language Model
LSTM	Long Short Term Memory
ML	Machine Learning

MLM	Masked Language Modeling
MLP	Multilayer Perceptron
MoE	Mixture of Experts
NER	Named Entity Recognition
NLP	Natural Language Processing
NMT	Neural Machine Translation
NSP	Next Sentence Prediction
OOV	Out of Vocabulary
RF	Random Forest
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized BERT pre-training Approach
SEO	Search Engine Optimization
SIF	Smooth Inverse Frequency
SoA	State of the Art
SVM	Support vector machines
TF-IDF	Term Frequency - Inverse Domain Frequency
TPLM	Transformer-based Pre-trained Language Model
VM	Virtual Machine

Chapter 1

Introduction

In this thesis, we discuss the problem of Entity Resolution (ER), with a focus on e-commerce. In the current chapter we introduce the full context of the problem being studied. We will also present here the processes our work follow, and the limits of the project's scope.

1.1 Problem

In table 1.1 we have two lists of TVs and we want to see if they contain matching pairs. Items on the left came from the website of Elgiganten [*] (data source A), whereas the items on the right came from the website of Webhallen [†] (data source B). *Our task is to identify offers in the two sources that refer to the same physical products*. One hint we can look at is the brand. Even when that information is not available in the dedicated column, we can identify it in the product name. Nevertheless, we can observe that multiple TVs are produced by the same brand so that can not be the sole signal. A stronger indicator is the model name, that is usually available for TVs. In this case we can see that Elgiganten's first product has the same brand and model name as Webhallen's second product. For humans it is easy to identify the model name just by looking at a product, but how would a computer do it? One way could be through matching a regular expression but this becomes difficult since there is no standard way of defining model names. If the Global Trade Item Number (GTIN) would be available, regular expressions would be a feasible solution, but retailers don't always make this information available.

[*] https://www.elgiganten.se/ [†] https://www.webhallen.com/

Product name	Brand	Price
TV OLED Lg OLED65G1 2021		4999
LG Electronics OLED55C1	LG	12000
TCL Tcl 65c825 65” Led-tv	TCL	7500

Product name	Brand	Price
LG Electronics 65NANO756PR	LG	8999
LG Electronics OLED65G1	LG	5198
Apple TV 4K 64 GB 2021	Apple	15000

Table 1.1: A sample of an ER problem (matches are highlighted)

Sometimes we do not have so much information available. Table 1.2 shows another sample of an ER problem. In this case we are taking a look at *entertaiment* products from two Swedish retailers. These are low cost, high volume products, and are usually less documented than high end products such as TV. We do not have a GTIN or even a model name for these products so the matching has to be purely textual. An ER model has to understand for example that a card holder with a flag model is different from the actual flag or that a card holder has little to do with card games or football cards. This example shows why language models are necessary if we want model to reliably interpret these subtleties.

Product name	Price
Football card Starter package Fifa 365 2021 (panini)	149
Ukrainian flag card holder	399
Nordic Uno basic card game family game	499

Product name	Price
Flag of Ukraine Ukrainian national flags Garden flags	159
Uno basic card game family game	549
UNO Minimalist card game - 2 to 10 players - 7 years and older	349

Table 1.2: A sample of a more difficult ER problem (matches are highlighted)

Simply put, the goal of any ER solution is to take two (or more) data sources and match those entries that refer to the same real-world data entity. A particular case, usually referred to as duplicate detection [1], appears when a data source is compared against itself.

In particular, in the case of e-commerce, the data sources are generally obtained from the catalogues of retailers, and their entries are called offers. The real-world entity these offers refer to are called products (sg. product). In other words, a pair of offers should be labelled as a match if and only if they advertise the same product.

As we move forward with the presentation of this work it is worth keeping in mind that the these are only samples. The text available for one single offer is usually much longer than what we could have shown in a report, and it can cover multiple functions (*i.e.,* title, description, specifications table). Moreover, the sizes of retailer's products catalogues are multiple orders of magnitude higher than the 3 offers per retailer we show in the two examples above. This dimensionality constitutes another important factor to keep in mind when designing a solution. The volume of data is also the main reason why we require an automatic solution.

1.1.1 Original problem and definition

We carry out this work under the supervision of engineers from Panprices AB, a Stockholm-based company that offers users the possibility to shop for products across borders. They continuously acquire offers from retailers across Europe. One challenge the face is to identify which of those offers refer to the same product. Panprices AB needs this information for displaying all available offers on a product's page to help their customer make the best decision when shopping online.

Their current solution for grouping offers together is using GTINs to identify products uniquely. The issue is that not all retailers make the GTIN available, so their offers may be left out. For the company, this means that they may fail to show their users the best offers for a product just because they cannot match them. Moreover, given the amount of data Panprices AB deals with, hiring humans to perform these matches is unfeasible. Some manual matching is performed for specific products in their catalogue, but these operations can not be extended to a large pool of products.

The company needs an autonomous tool that would take as an input the catalogues of offers and give matching pairs at the output. It is important to the company that false positives are rare (measured by False Positive Rate (FPR)), while false negatives are more admissible (measured by False Negative Rate

(FNR)). In other words, it is better to overlook some offers than to add offers to the wrong product list. That is because a customer's experience is more negatively influenced by a false announcement than a lack of one offer for a given product. Panprices have defined that an ideal range for their operations is FPR $< 1\%$ and FNR $< 5\%$

1.1.2 Scientific and engineering issues

From a scientific perspective this is an interesting problem due to the difficulties it introduces. As presented in Section 2.1, this is an ER problem, which has been studied for a long time. Various methods have been applied to the problem: probabilistic methods [2], evolutionary algorithms [3], Machine Learning (ML) [4], Recurrent Neural Networks (RNNs) [5]. Lately, following the success of Transformer based architectures [6] on Natural Language Processing (NLP) tasks, we have seen proposed solutions that are powered by Transformer-based Pre-trained Language Models (TPLMs) [7, 8].

Since the current state of the art is given by TPLM based solutions [7, 9], we will focus on them. At the same time, we will have a look at some information about earlier alternatives for comparison purposes.
It has been shown [7, 8, 10] that these models are more effective than their predecessors, especially when a large dataset is available and when the data is less structured, has "dirty" fields or missing values.

1.2 Research question

We address the following research questions:

Can we use Transformer-based Pre-trained Language Models (TPLMs) to automatically identify e-commerce offers concerning the same product? What is the correlation between model performance and training data volume for such tasks? How can TPLMs improve the task of ER for e-commerce when low data volumes are available?

1.3 Purpose

Our purpose in this project is to identify, select, improve and use State of the Art (SoA) solutions for ER. We are interested in two particular contexts:

1. The first context we consider is the one where it is not possible to automatically generate a dataset for solving the ER problem. Particularly, we want to know how various solutions deal with situations where the trainig data is scarce.
2. The second context we consider is the one where a dataset can be automatically generated by using GTIN heuristics.

1.4 Goals

First and foremost, the goal of the thesis project is to understand the landscape of solutions already available in the literature. How do they differ? What similarities can one identify among some / all of them? Are the solutions domain-specific (e-commerce, people registry, health domain), or are they more generally applicable?

Secondly, we will identify ways in which we can improve the performance of SoA models. We apply our ideas and demonstrate their usefulness through experiments.

Last but not least, we want the solution to be directly applicable to a need in the industry. Therefore, the generated knowledge should be transferable to the environment Panprices AB is dealing with. More precisely, this means that we are looking at the performance of various solutions on the company's proprietary data, as well as openly available datasets.

Our contributions are:

- we present *Spring R-SupCon*, an improvement on the SoA (Goal 2)
- we demonstrate the power of smaller language models in the context of Entity Resolution (ER) (Goal 2)
- we introduce a way to produce lower volume datasets from open datasets to diversify the conditions under which Entity Resolution (ER) models are tested (Goal 2)

- we provide a review of the ER scientific literature (Goal 1)
- we apply the gained knowledge to the particular case of Panprices AB (Goal 3)

1.5 Research Methodology

As stated in the previous section, we are looking at solutions for the ER problem. These solutions are in fact *models* that try to understand whether a pair of offers is describing the same product or not. In other words, our goal is to *model* the process through each a human is able to tell when two offers are a match.

We conduct our work with the objective of obtaining *quantifiable* results. Through our methods, we perform a quantitative analysis of various solutions from the SoA, through *experimenting*. Our results are generated by *observing* the output of the models we study. According to data science best practices, we split all of the studied datasets in three partitions: training set, validation set and test set. We test all the models on the same datasets and we measure their performance using the same *operationalization* methods to isolate the model's ability to understand the problem as the sole generator of differences in results.

For the goal of comparing between different models, we are not interested in particular examples. We use operationalizations of the model's performance that take into account the *statistical* results observed when testing the model against the true results.

1.6 Delimitations

We will see in Section 2.1, that an ER solution may choose to focus on one of the two major components: blocking or matching. The blocking algorithm has been thoroughly explored in the literature [11, 12, 13]. Even if it has a major impact on the functioning of the whole ER pipeline, we choose not to focus on this part. Instead, we will use the same blocking algorithm for all the matching algorithms to eliminate differences that could arise from this step.

Another area where the literature focuses is schema alignment [14, 15, 16]. We purposefully overlook this step because Language Models (LMs) are inherently schema-agnostic. All the information related to one offer is concatenated into a single string [10, 8]. Where the authors suggest [7], we use the attribute names as well and use additional tokens for distinguishing attribute names and values.

1.7 Sustainability and ethics

The data we use for this project is either coming from open datasets or from the host company's catalogue. None of the data we use is targeting individuals, and as such it does not contain any sensitive information protected by privacy laws such as General Data Protection Regulation (GDPR). We target a problem that is too large to be manually handled by humans, so the automation methods we present could not be used to replace humans, but rather to enhance their potential in navigating the e-commerce space. Therefore we consider that no *ethical* issues arise due to the current work.

For the goal of generating the concrete results of this work, we run experiments in the cloud. Because of that, our energy consumption is strictly correlated with the amount of money we spend, so we are motivated to use as few resources as possible. We provide an automated script to make sure that the Virtual Machines (VMs) we use do not run for longer than required, saving energy when possible. In our work, we also address the analyze the benefits of larger models over smaller ones, taking into account the correlation between energy consumption and observed performance. We deem our project *sustainable* in the virtue of these traits.

1.8 Outline

In Chapter 2 we present a survey on the current ER solutions. We also survey LMs and explore ideas that might be relevant for the current problem. In Chapter 3 we discuss the main contributions of this project to the scientific literature, as well as how the problem of ER for e-commerce can be approached in the industry. We present and discuss the results of this work in Chapter 4. Finally, we give our conclusion in Chapter 5, as well as directions that may be worth following in future research.

Chapter 2

Background

We will begin this chapter by looking at a general architecture of an ER solution. Secondly, we will take a look into what TPLMs are available, how they relate and how they differ from each other.

Additionally, we will look at different approaches to solving the ER problem, including both classical approaches and transformer-based solutions. The purpose is to understand whether previous research confirms the intuition that TPLMs have pushed the performance for ER solutions. If so, how are they integrated into the overall architecture of the more modern solutions?

2.1 General architecture of an ER solution

The ER problem has been studied for a long time [17], and new solutions have been generated as Computer Science (and its sub-fields) evolved. Over the years, it has been known under different names such as entity matching, data matching, duplicate detection, or data linkage [1].

The most straightforward architecture of a possible solution is shown in Figure 2.1. Let $a \in A$ be an entry in data source A, and $b \in B$, an entry in data source B. The job of the matching block is to look at all pairs $(a, b) \in A \times B$, belonging to the cartesian product of the two sets (data sources) and decide which of them refer to the same real-word entity.

One obvious problem with this approach is that the matching block will have to go through all the possible pairs $(a, b) \in A \times B$. For $|A| = m$ and

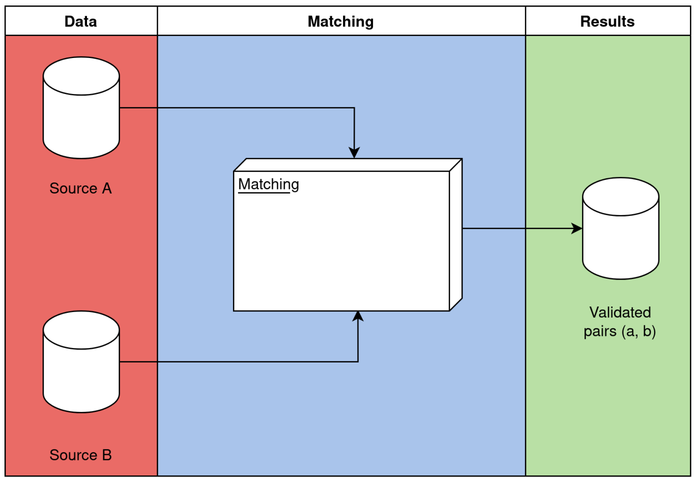


Figure 2.1: Simple architecture for an ER solution

$|B| = n$, the algorithm will have a complexity of $O(m \times n \times p)$, where p is the computation required for checking one potential match. For example, if both retailers have 100,000 products in their catalogue and it takes 1ms to check one pair, then checking all the possible pairs would take 115 days. We can reduce the inference time by ensuring a rapid assessment of every possible pair (low values of p). We can do this in a very simplistic way by reducing the information we process, only using the input signals we consider the most relevant. Another way we could gain speed is by reducing the number of parameters and the calculations performed by the model, but both of these may affect the accuracy of the predictions. Instead, we are looking for a fast way to discard unfeasible pairs, while keeping the accuracy high for pairs where the decision is more difficult. To achieve this, we introduce an additional step, called blocking, to discard unfeasible pairs, as seen in Figure 2.2.

Leveraging recent progress in the fields of Machine Learning (ML) and Deep Learning (DL), newer solutions [4, 5, 7, 8] train a matcher block to reliably verify pairs, using massive datasets. The ML architecture for the ER task is shown in Figure 2.3.

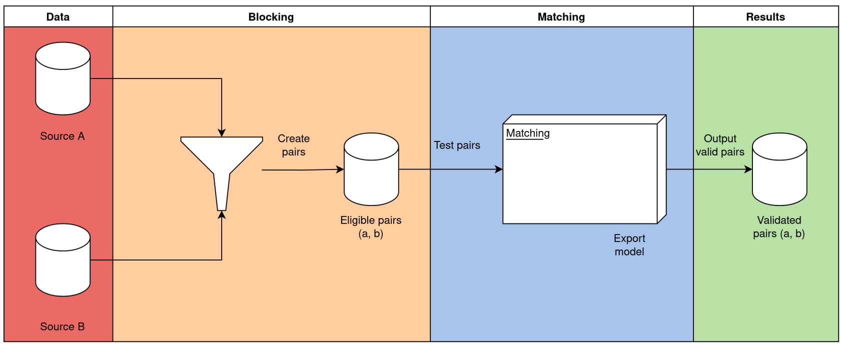


Figure 2.2: Architecture for an ER solution, including a blocking step

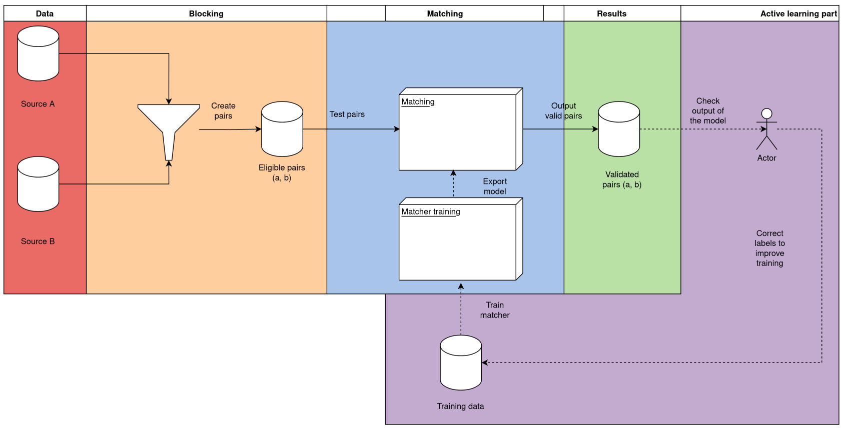


Figure 2.3: ML architecture for an ER solution, including a blocking step

2.2 TPLMs

2.2.1 The Attention Mechanism

The Transformer architecture, introduced by Vaswani et al. [6], took over NLP, showing better performance in a wide range of tasks. The original Transformer consisted of several encoding layers stacked upon each other and corresponding decoding layers. It was designed for a task of Neural Machine Translation (NMT). The great advantage it has over the previous SoA, RNNs, is that it is highly parallelizable, taking advantage of modern hardware, namely Graphical Processing Units (GPUs). Another great advantage of this architecture is that it can produce context-aware embeddings instead of static embeddings like GloVe [18], word2Vec [19] or fastText [20].

Multi-headed attention mechanisms power up every encoding and decoding layer [6]. To understand the previous phrase, we first need to understand what attention mechanisms are. This concept is older than the Transformer architecture and has been previously used with RNNs to mitigate the issue of losing connections over a long sequence. In a nutshell, having an attention mechanism means that we attribute some weight to all of the words in a sentence (the *context*) when we compute the embedding of one *target* word. This is illustrated in Figure 2.4, where we have the word "bank" as *target*, and we are trying to create a *contextualized embedding* for it. The context is important because the same word can have different meanings depending on the sentence where it is used. We see this with the word "bank" here, and we saw it with the word "card" in Table 1.2.

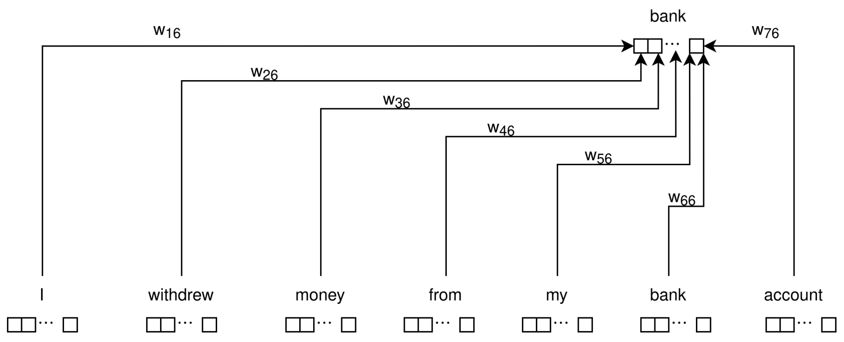


Figure 2.4: The intuition behind an attention mechanism

Computers are not able to understand words as humans do. To overcome this challenge, words are embedded in a high dimensional space. In other words, a vector is attributed to each word. The more semantically similar the words are, the closer they should be in this space. Thus, a computer can make sense of the meanings behind the strings. This is the way in which word2vec and GloVe operate, but it is problematic when we encounter homonyms, words with the same form that hold different meanings. Humans can understand that the word "bank" in the sentence above refers to a financial institution, as opposed to an object on which one can sit, based on the context provided by the other words.

To account for context, each word j takes from the meaning of the other words. In the figure above, $w_{i,j}; i, j \in \{1, 2, ..., 7\}$ represents the weight attributed to word i when calculating the embedding for word j. Finally, the new embedding of word j becomes $v_j = \sum_{i=1}^{7} w_{i,j} * v_i$. This means that not all the words in a target's context are considered equal. For example, we can expect the words "money", "bank" and "account" to carry the most relevant meaning to the *target* word "bank" in this context. Thus, we expect w_{36}, w_{66} and w_{67} to have higher values than those of the others.

These weights ($w_{i,j}$) are calculated by looking at the similarity in meaning of every pair of words. More precisely, Figure 2.5 details how the weights are calculated for the target word "bank".

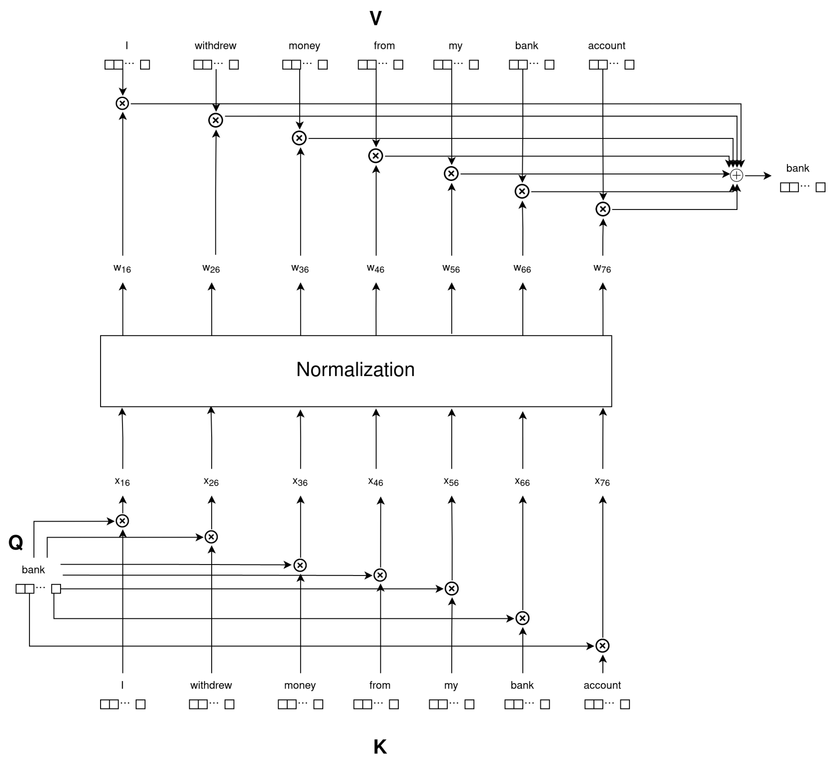


Figure 2.5: Detailed representation of an attention mechanism

In Figure 2.5, the attention mechanism takes the vector representation of the word "bank" and performs a dot product operation between it and the representation of all the words in the sentence. This gives the model a sense of how similar the words are, which then indicates how important each word in the *context* is for understanding the meaning of the *target*. The results are normalized. This prevents the representations from becoming larger with each summation. Finally, the upper part of the figure depicts the same process as shown in Figure 2.4, where each embedding is multiplied with a weight and then all the values are added up to form the new representation of the *target* word “bank”.

Notice that during the attention process, the representation of a word is used 3 times, marked on the figure with letters *K, Q* and *V*. These letters stand for key, query and respectively value. They have been inspired from a database analogy. The *target* word is the query, and we try to match that against all the keys through dot products. In a database use case this should return a sample of values whose keys are matching the given query. Instead, each value is considered to match to some degree the given query, but the values are scaled by the similarity between query and key to account for importance.

To distinguish between the three different representations of words, the model uses three matrices: M_Q, M_K, M_V. A word is transformed using one of the matrices to obtain a representation tailored to one of the three scenarios (query, key or value). For example, to obtain the query representation of the word “bank”, with embedding $v_6 \in \mathbb{R}^d$, we simply perform $q_6 = v_6 * M_Q$. The representation of the word “bank” will be different if it is used at the bottom (as a key), as the target word (query) or at the top for finally computing the new representation (as a value). These matrices contain the parameters being learned during the training of attention-based models.

So far we presented how an attention mechanism works. Instead of only one such mechanism, a Transformer hosts multiple *attention heads* per layer, thus the name of “multi-headed attention mechanism”. Each attention head is initialized differently, and learns to look for different relations between words [21].

2.2.2 An overview of LMs

Based on the paper published by Vaswani *et al.,* [6], the idea of having pre-trained LMs for obtaining contextualized embeddings for other NLP tasks came to life. These LMs usually contain either just the encoder or just the decoder portion of the Transformer architecture. In this subsection, we will take a look at different TPLMs publicly available.

Embeddings from Language Models (ELMo) [22] was one of the first models to propose contextualized embeddings in the field of NLP. Their architecture was not based on Transformers, and the resulting embeddings were to be used without the opportunity to fine-tune them for a downstream task. Nevertheless, ELMo deserves to be mentioned as one of the pioneers of LMs.

Generative Pre-training Transformer (GPT) [23] was the pioneer of Transformer-based Pre-trained Language Models (TPLMs). Researchers from OpenAI demonstrated how their model could achieve State of the Art (SoA) results in popular tasks in the field of NLP, such as Natural language inference, Question Answering, Sentence Similarity and Classification. Their architecture is based on the decoder part of the original Transformer architecture. The model was pre-trained on a "diverse corpus of unlabeled text". The authors published the model alongside the paper, enabling anyone to use it to solve other problems where an understanding of language was beneficial or even necessary. The same research group gave us some evolutions of this model in the forms of GPT-2 [24] and GPT-3 [25].

Bidirectional Encoder Representations from Transformer (BERT) [26], introduced by Google in 2019, is probably one of the most spread TPLMs. It and its variants are cited in most of the newer works focusing on NLP tasks. As we will see in Section 2.3, solutions for the ER problem benefit from using this TPLM as well. The novelty brought by BERT is its bidirectionality, which was not present in the initial transformer architecture [6]. For training BERT, Devlin *et al.,* developed an algorithm to mask some of the tokens using a special "[MASK]" token and then have the model predict what those tokens could be. This process is known as Masked Language Modeling (MLM). A secondary task used for training BERT is Next Sentence Prediction (NSP), a task in which the model is given two sentences to predict whether the second one can be following the first one in a corpus from a logical perspective. The data this TPLM has been trained on is composed of books and the English Wikipedia. The authors released the base model and other versions obtained by varying the number of parameters in the model.

Later, Yang *et al.,* [27] argued that using the "[MASK]" token in pretraining BERT creates a discrepancy between pretraining and fine-tuning BERT for downstream tasks. Instead, they proposed another model called XLNet, which is autoregressive. To tackle the issue signalled by Devlin *et al.,*

[26], that language is inherently not autoregressive, the authors of XLNet [27] came up with a permutation algorithm. The main idea was that the model would see multiple permutations of the same sequence of tokens, thus using the information of subsequent tokens (that appear before the target in at least one permutation). Their experiments empirically prove that XLNet performs better than BERT in all the downstream tasks presented in the original BERT paper.

Liu *et al.,* [28] argued that the architecture and training objectives proposed by XLNet are not, in fact, superior to the ones used by BERT. Instead, they show that XLNet was trained on a more extensive corpus than BERT. Thus, the performance improvement comes from the model seeing more data during training time rather than from a superior architecture. Robustly Optimized BERT pre-training Approach (RoBERTa) has, in fact, the same architecture as BERT, but it was trained on a larger corpus. Additionally, the Next Sentence Prediction (NSP) task is removed from pretraining. RoBERTa's authors empirically prove that it performs better than XLNet, demonstrating that their hypothesis is true.

One big issue of TPLMs is that they require many parameters to be trained, making them resource hungry [29]. There are researchers [29, 30, 31] who proposed ways to reduce these models while keeping the performance close to the bigger models. A Lite BERT (ALBERT) [30], for example, proposes two techniques for reducing the number of parameters: decoupling the embedding size of the output from the embedding size of the hidden layers and parameter sharing across the encoding layers. The former means that the size of the embeddings in the hidden layers can grow, allowing them to capture more information but without inflicting a quadratic increase in the number of parameters. The latter reduces the parameters used by a factor equal to the number of encoding layers used.

Another work, DistilBERT [29], proposes reducing the number of parameters in the model by using fewer encoding layers. In order to keep the performance of the model high, they perform knowledge distillation, allowing the smaller model (student) to learn how a bigger model (the teacher) responds to different inputs. The student model is thus, able to learn how the teacher model processes information and learns to replicate the results using a smaller number of parameters.

Similarly to DistilBERT [29], Jiao *et al.,* [31] propose a distillation algorithm to fit the power of BERT-base into a smaller model. What makes TinyBERT different is that the distillation learning is not only applied to the last layer but information from intermediate layers is also used to train the student model. Two additional loss function constituents are added to this end: hidden state distillation and embedding layer distillation—the former aims to reduce the difference in the attention matrices between the teacher and the model, and the latter aims to reduce the difference in the hidden layer embeddings themselves. These two members are added to the prediction layer distillation loss (also used by DistilBERT) to form the overall loss function for training TinyBERT.

2.3 A survey of the ER landscape

As mentioned in Section 2.1, the problem of ER has been studied for a long time [17]. Naturally, since then, many solutions have been proposed, but we limit the scope of the current survey to the last few years when ML and DL methods were employed. We categorize the existing solutions based on the core technology they used, which is also strongly correlated to when the solution was published. We will first look at solutions created before the popularity wave of DL algorithms. Next, we move on to solutions using a DL architecture, but not a TPLM. Last, we take a look at the ER solutions at whose core stands a TPLM.

2.3.1 Traditional ML

In this category, we look at one particular solution, Magellan [4]. This had been the SoA in ER before the increase in popularity of DL architectures. It is a flexible solution, interfacing easily with all the tools from the Python data environment (e.g. pandas, NumPy). It implements ready to use blocking methods, as well as matching methods based on well known ML methods: Decision Trees (DTs), Random Forest (RF), Support vector machines (SVM), XGBoost, etc. Features are manually engineered for this method, using Jaccard Similarity [32], Levenshtein distance [33], cosine similarity [34], or other classical strategies applied on different n-grams (e.g. 3-grams, 5-grams). One issue with this approach is that the two data sources being compared need to have their schemas aligned which can be an issue, especially when comparing offers promoted by different retailers, each using different attributes. Another obvious issue representative of this type of approach is that it requires a lot of engineering effort to develop the right features to use.

Considering the completeness of this solution and the fact that it was unanimously recognized as the SoA in subsequent papers, we decided not to look at more examples from this category.

2.3.2 DL based methods, before Transformers

Most of the solutions in this category belong to the subcategory of RNN-based architectures. Their popularity does not come as a surprise since RNNs represented the general paradigm of working with NLP before the advent of Transformers. These networks are well suited for handling sequences, and texts are sequences of tokens (words or other types of tokens such as n-grams). Their power stems from the fact that they can take into account previous tokens when looking at the current one, enabling the model to establish relations among them. One issue with RNNs is that they are not able to identify connections between tokens that are far apart in the sequence. Based on the RNN idea, other improvements were made to tackle this issue, in the form of Long Short Term Memory (LSTM) mechanisms and Gated Recurrent Units (GRUs). In the context of ER, names are sequential as shown in Table 1.2. Another advantage of using RNNs is that different attributes can be serialized in the same string when a specification table is available. This enables us to use unstructured information without the need for schema alignment.

Moreover, RNNs are autoregressive by nature, but language is not. A token's neighbours give its context on either side. For example, in the sentence "I withdrew money from my bank account", looking at the token "bank", we can see that the tokens "money" and "account" both provide relevant information about the target. Therefore, bidirectionality was added [35], resulting in Bidirectional LSTM (BiLSTM) and Bidirectional GRU (BiGRU). As seen in Section 2.2, the major drawback of these architectures remains the fact that they are sequential and, thus, cannot take advantage of hardware optimized for parallel computations.

The first solution from this category that we discuss is DeepMatcher [5]. This project comes from the same research group as the previous state of the art, Magellan [4]. In fact the DeepMatcher paper [5] proposes four different DL solutions for the problem of ER, namely: Smooth Inverse Frequency (SIF), RNN, Attention and Hybrid. We will focus our analysis on the Hybrid model, the most complex, achieving generally better results, which was subsequently referenced as SoA in DL for ER. The idea behind this architecture is to combine BiGRUs with an attention mechanism. One major advantage of this method over Magellan [4] is that the schema of the tables being compared does not have to be aligned anymore. All the attributes describing one offer can be concatenated in a sequence and fed to this network. It uses the fastText library for token embedding. One difficulty in using this model, similar to all the others using RNNs is the impossibility of using modern hardware to perform computation in parallel.

The authors of DeepER [36] also focused on an RNN based solution. Similarly to DeepMatcher [5], the attributes are individually embedded through the use of RNN. There are also substantial differences. One is the use of LSTM networks instead of BiGRUs. A second one is that this model requires schema alignment because it considers all the attributes as separate sequences of tokens and then performs similarity checking on corresponding attributes. This limitation should ideally be avoided since we are looking at comparing catalogues from different retailers which will not adhere to the same schema. A notable difference is also the use of GloVe word embeddings instead of using the fastText library. This is relevant, since GloVe [18] focuses on tokens, whereas fastText [20] focuses on character level embeddings. This is a trade-off discussed by Mudgal *et al.,* [5]: word level embeddings capture more meaning but are more sensible to Out of Vocabulary (OOV) words compared to character level embedding techniques. Ebraheem *et al.,* [36] thoroughly discuss blocking methods as well, offering a complete pipeline for ER, but those are out of the scope of the current project.

Hi-EM [37] is another solution based on RNNs. Nevertheless, the architecture it proposes is quite different from the previous models. Zhao *et al.,* [37] did not use any embedding library or pre-computed word embeddings. Instead, character level embeddings are learned by the model at training time. Another big idea this paper brings to the table is the use of Knowledge Bases (KBs) for identifying corresponding columns in different schemas. The central idea is using KBs published by search engine operators, in this case, Microsoft, for performing a kind of Named Entity Recognition (NER) and base attribute mapping and attribute comparison on that information. Because it does not use any embedding library, all the model needs to be trained from scratch, limiting the ability to reuse previous work, and requiring larger datasets and longer training times.

The last RNN based model we look at is PMM [38], or Product Matching Model. Li *et al.,* used Convolutional Neural Networks (CNNs) as well for mapping the interaction between the words of the two different titles being compared. The idea is to take the word embeddings of the two titles (initialized by Word2Vec and processed by a BiLSTMs layer) and multiply the matrix to obtain a matrix of word interaction. A CNN looks at this matrix of word interaction to extract features to pass to a final Multilayer Perceptron (MLP) step. The authors describe another component of PMM that also uses attributes. This other component is quite similar to the one matching the titles we just discussed, albeit more complicated to consider minor differences in attributes. The base components still stay the same. We will not go through this second component in greater detail. Because this model uses Word2Vec to create word embeddings, it is not able to handle OOV words. Often ER data contains product model tokens as seen in Table 1.2, which are not likely to be part of a training vocabulary. Not being able to use such a strong signal represents a big issue for this model.

Wang *et al.,* [39] introduced a model called CorDEL. In their paper, they observed a pattern in all the DL solutions for ER, namely that they are all based on siamese networks [40]. In other words, titles (and attributes) describing an offer are independently mapped to an embedding space by identical networks. Then these embeddings are compared, and a similarity score is calculated. In their paper [39], they make the argument that such a way of performing ER misses out on some information regarding word-level similarities between the two offers. Thus, they propose a contrastive approach, looking at the shared words and the words that separate the two offers for making the decision. They use the fastText library for word embedding, and they process the inputs using Feed Forward Neural Networks (FFNNs) with dense connections. This solution drastically reduces the number of parameters trained in the network as compared to RNN based solutions like DeepMatcher [5] while showing performance which is on par with and even exceeds that of the previous SoA. While it takes into account the contrast between the two offers, this method misses the context in which the words are used, following a Bag of Words (BoW) approach. This means that CorDEL would consider the words “flag” or “cards” in Table 1.2 to be exactly the same regardless of their respective contexts.

2.3.3 Solutions based on TPLMs

One of the first ER models to use TPLMs was Ditto [7]. Li *et al.,* leverage the power of BERT, to which they only add a classification layer which takes as input the embedding of the “[CLS]” token. This architecture is very similar to another contemporary work [10], but Ditto adds a few optimizations as well. Moreover, in the Ditto paper we can find a proof that there was an issue with the paper redacted by Brunner *et al.,* [10]. Namely, they did not use a validation set, optimizing their hyperparameters directly on the test set. Schema alignment, or misalignment thereof, is not an issue for Ditto, or other subsequent Transformer based solutions, because, as seen with DeepMatcher [5], all attributes can be concatenated into one long string, which is then fed to the LM. As mentioned, Ditto [7] can apply a series of optimizations: leveraging domain knowledge, summarizing long entries, and augmenting training data. For more details, the reader is referred to the original paper [7]. The issue with Ditto is that it focuses on pairs, instead of independently embedding each offer. This means that the model has to be called multiple times for the same offer depending on the pair in which it is present.

Peeters *et al.,* [41] propose a very similar approach where pairs of offers are embedded together by a TPLM. Their main contribution stems from the fact that they emphasize how TPLMs are trained on general corpora of text from Wikipedia and the news, which is somehow different to what the model sees in the task of ER for e-commerce. Therefore they propose to further adapt the LMs to the domain of e-commerce using vast amounts of product offers data, with the Masked Language Modeling (MLM) objective originally used in training BERT [26]. As discussed for Ditto, the challenge here is that the model has to be called multiple times for the same offer, because it’s embedding is also dependent on the pair to which it belongs.

The creators of eComBERT [42] propose a new training objective, similar to the one introduced by Sentence-BERT [43]. Instead of training their model on tuples, they create a triplet where the product name serves as an anchor. The following two elements are a matching and a non-matching offer for the product. The objective of the training is to get the matching offer close to the anchor product in the embedding space while increasing the gap between the product and the negative example. Compared to the previous solutions, eComBERT uses a siamese network, being able to reuse the embeddings of each individual offers. On the other hand, it misses the opportunity to look at words belonging to different offers, suffering from the issue that presented by Wang *et al.,* [39].

JointBERT [8] is another solution that stands out with a different learning objective than the norm. In this paper, Peeters *et al.,* propose a dual objective for the model. A classification layer that aims to assign the two offers being compared to a known GTIN (product) is added. They demonstrated that this method yields better results when offers present in the test set refer to products present in the training set. However, it performs poorly on offers referencing completely new products. Additionally, the Peeters *et al.,* use interpretability methods 2.2 to demonstrate the pieces of information that their model considers the most important when making a decision. Although the model shows improved performance for products whose GTINs it has seen during training, the performance for new products decreases.

Jain *et al.,* [44] shifted the perspective, looking at ER from an active learning angle. They laid out the process, including a human in the loop, who would verify the model's output (for cases with low confidence) and thus generate new training data for improving the model. Figure 2.3 also shows this part of the pipeline by the connection going from the user back to the dataset. The novelty introduced by the paper stems from the introduction of a sampling method for looking at the most relevant examples (which is highly desirable when humans are included in the loop), as well as a way to train a blocker based on TPLMs. The matcher they used is very similar to what the previous papers describe. While the other contributions this project introduces are significant for the field of ER, they are out of the scope of the current project, so we will not discuss them further.

One of the latest solutions for the problem of ER, as of the time of this writing, is presented by DAME [45]. Trabelsi *et al.,* based their architecture on the idea of creating Mixture of Experts (MoE) layer. In other words, a layer connected at the top balances the suggestions of each *expert* through an attention mechanism. The main idea behind this proposal is to take advantage of all the data available for training ER solutions, independently of the target domain. Thus, multiple, parallel *experts* are trained, each specialized in one type of product and then using this MoE a good generalization comes out of the diverse perspectives. The paper empirically shows that their solution highly outperforms Ditto [7] especially when data for training on a specific category is missing or insufficient for generalization. Even when higher amounts of data are available, DAME is either on par or slightly better than its counterpart. To minimize the impact of fine-tuning multiple LMs in parallel, hence tweaking a significant number of parameters, the authors use DistilBERT, which is a lighter version of BERT, as explained in Section 2.2.

Finally, in their most recent paper, Peeters *et al.,* [9] propose a contrastive method for ER inspired by the success of such approaches [46] proved to have in the domain of Computer Vision (CV). The contrastive objective comes as a pretraining step. This method aims to look at all offers within a batch and bring the ones referencing the same product closer while maximizing the distance among offers targeting different products within the embedding space. This project introduces a data sampling as well in order to avoid misleading labels during this pretraining phase. After this step, the model is fine-tuned with the usual Tuple Classification objective for predicting if a tuple of offers is targeting the same product or not.

2.4 Datasets

A critical aspect of research that could not be included in the previous two sections is the creation of datasets to train and test all of these solutions. Some of the datasets used were done manually, hindering their size. The WDC dataset introduced by Peeters *et al.,* [47, 48] introduced a new method of creating large datasets automatically. Most e-commerce websites annotate their product pages, adhering to the recommendations of schema.org for Search Engine Optimization (SEO) purposes. A lot of these annotated pages contain a GTIN as well, so one can quickly build a dataset by looking for the same GTIN offered by different vendors. This method of creating a dataset is an example of distant supervised learning, where the model is trained using labelled data. However, this data is automatically generated from sources on the internet. One thing to pay attention to when using this method is that the labels may not be all the time accurate, GTINs may be absent or wrongly assigned. Nevertheless, it has been shown [48] that data obtained as such is good enough for training ER systems. Peeters *et al.,* published a manually curated *golden standard* as well for confidently testing the results models.

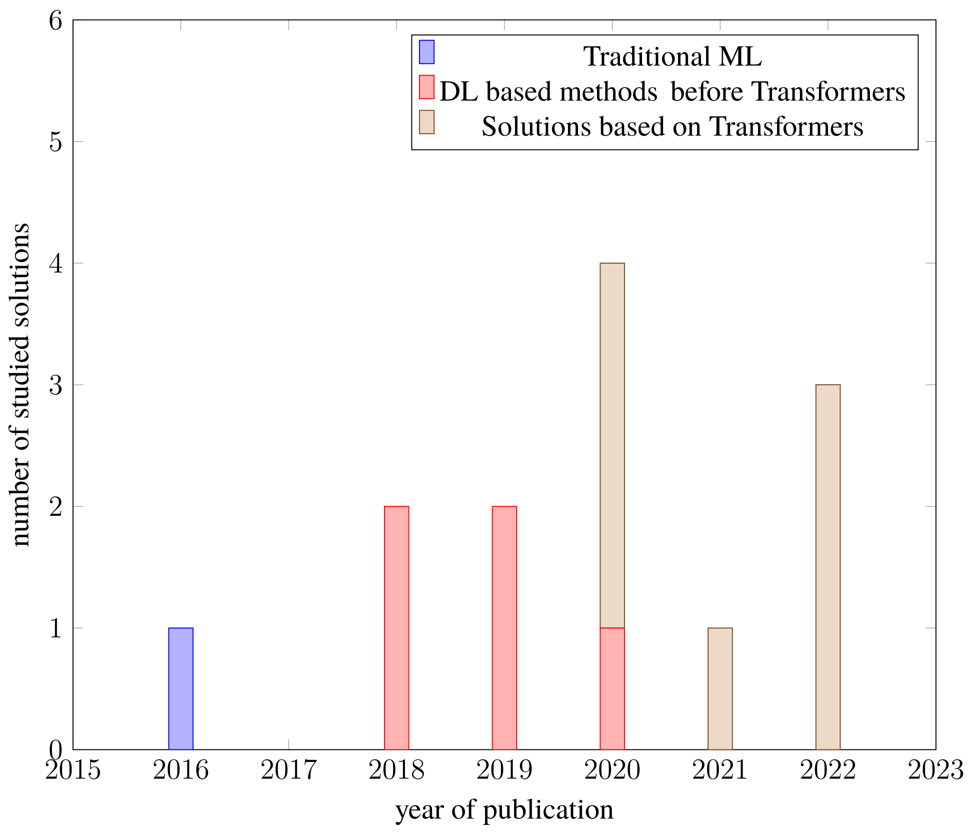


Figure 2.6: Evolution of algorithms being used for solving the ER problem

2.5 Summary

We began this chapter by taking a good look at the Transformer architecture and the LMs based on it. In the second part of the chapter, we focused on specific solutions to the problem of ER. We saw how TPLMs improved the SoA for this task, as it has happened for other NLP tasks. Finally, we saw how distant supervised learning can produce massive ER datasets.

A summary of the reviewed solutions can be found in Table 2.1. Figure 2.6 shows the evolution of ER solutions throughout the years. It is worth remarking how distinct *eras* can be identified when looking at ER solutions, with Transformers dominating the field in recent years.

Solution	Technology					Training objective			
	ML	FFNNs	RNNs	CNNs	Transformers	Tuple	MLM	Triplet	Contrastive
Magellan [4]	✓					✓			
DeepMatcher [5]			✓			✓			
DeepER [36]			✓			✓			
Hi-EM [37]			✓			✓			
PMM [38]			✓	✓		✓			
CorDEL [39]		✓				✓			
Ditto [7]					✓	✓			
Peeters *et al.*, [41]					✓	✓	✓		
eComBERT [42]					✓		✓	✓	
JointBERT [8]					✓	✓			
DIAL [44]					✓	✓			
DAME [45]					✓	✓			
R-SupCon [9]					✓	✓			✓

Table 2.1: Analysis of previously proposed solutions from the literature

Chapter 3

Methods

In this chapter we address the steps taken for answering the research questions presented in Section 1.2. We discuss here the metric used for model comparison (Section 3.1), dataset selection / creation and discovery (Sections 3.2 and 3.3), implemenentation details of two models from SoA (R-SupCon [9] and Ditto [7]) (Section 3.4), the idea behind our model (*Spring R-SupCon*) (Section 3.5), our method for generating lower volume datasets (Section 3.6) and the effects of using smaller TPLMs with R-SupCon (Section 3.7).

3.1 Identifying an evaluation metric

As discussed in Section 1.5, identifying an evaluation metric is crucial for being able to quantify the performance of identified models. In other words, the evaluation metric is the basis for operationalizing model performance.

Before choosing such a metric we need to understand the kind of model we are confronted with. In Section 2.1 we understood that the problem of ER can be described as a binary classification problem where the model receives a pair of offers, compares them and outputs the probability of the pair belonging to a "matching" class (or to the "not-matching" class otherwise).

Classification problems have been thoroughly studied in the literature previously, so there is an abundance of already well established metrics one can use. On the web*, one can easily find articles explaining the most popular

* https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

metrics for this task such as: accuracy, precision, recall, Area Under the Curve (AUC), F1 score, F-beta score.

Metric	Advantages	Disadvantages
Accuracy	Simple	Not suitable because of dataset imbalance.
Precision	Easy to correlate with the business need.	Does not take false negatives into account.
Recall	Easy to correlate with the business need.	Does not take false positives into account.
F1-score	Represents a good aggregation of precision and recall.	It doesn't take into account that having false negative is preferred over false positives.
	It has been used in all of the studied solutions from SoA.	
AUC	Can help us choose a good threshold	Too broad, not well fit to the studied problem. Considering the use of softmax activation in the last layer, the probabilities are either very high or very low.
F-beta score	Takes both precision and recall into account. Enables the balance of the number of false positives vs. the number of false negatives.	Introduces an additional hyperparameter. It is not used by SoA, so it makes it difficult to compare the performance with existing work.

Table 3.1: Analysis of available metrics for classification tasks

Table 3.1 shows a summary of the advantages and disadvantages presented by various metrics for classification tasks. Looking at the table, we can clearly see that metrics like accuracy, precision, recall or AUC are not feasible for the ER problem (at least not as the main metrics). The discussion between

F1-score and F-beta score becomes more complicated, with strong arguments for both. For the purpose of the current project, we took the decision to continue with F1-score as main metric, mainly since it allows easy comparison with other solutions and it takes into account both false negatives and false positives.

3.2 Datasets

3.2.1 Open dataset selection

Multiple datasets have been created for testing ER models, each with its particularities. In this project, we will study three open datasets: `abt_buy`, `amazon_google` and `wdc_computers_medium`. All three of them have previously been used to test ER models [7, 9]. Along with the data itself, all three selected datasets are distributed with a standard train, test, and validation split, making it easier to reproduce and compare results obtained by different models.

3.2.2 Proprietary dataset creation

The proprietary dataset is trickier by nature. The host organization cannot provide a labelled matching dataset, but they store data for many individual offers from various retailers across the continent. As suggested by the authors of [48], this can constitute the base for a distant supervised labelled dataset. Namely, each of these offers identifies the target product by the GTIN, which is a strong signal for identifying a matching pair, while offers targeting different GTINs can be considered as non-matching with high confidence. Since this is a complex process, we dedicated the rest of this subsection to understanding how the proprietary dataset is created.

Table 3.2 shows the schema of offers as they are registered in the company's database. True positives can be inferred using GTIN (*product_id*) correlation, but sampling non-matching examples is also required for training a Deep Neural Network (DNN) model. A few questions still arise: *which kind of products should be selected?*, *should we use all positive matches?*, *how do we select non-matching examples?*.

Field name	**Field description**
offer_id	Unique identifier of the offer. An offer represents the listing of a product on a given retailer's website.
product_id	Identifier of the product. A product represents the actual thing being sold, so this ID stays the same across different retailers.
country	The country in which the retailer which proposed the offer activates.
retail_prod_ name	The name of product as described by the retailer.
price	The price at which the product is being sold expressed in the currency described by the 'currency' column.
currency	The currency in which the price is expressed.
offer_source	The source where the offer originated from. That is different from the retailer. A retailer can be present in multiple source, and a source will have multiple retailers.
retailer_name	The name of the offering retailer.
metadata	Field containing JSON with additional information such as: category of the product, images URLs and technical specification.

Table 3.2: The schema of the offers stored by panprices

A specific category, namely TVs, was chosen to sample data from for the proprietary dataset. Starting with a list of TVs provided by the company, we identified all possible matching offers using the *product_id*. As demonstrated by previous works [48], product offers sharing the same *product_id* can be considered as matching. Thus, a set of positive labels is generated through this first step.

Next, we have to generate negative labels. The following negative label generation heuristics are used to ensure the model is trained and evaluated in a variety of different situations:

- Pairs that are hard to label as a non-match
- Pairs that are easy to label as a non-match
- Pairs of offers belonging to the same brand that are non-matches

While the process for obtaining the last category is obvious, the other two categories require further explanations. For identifying the first two categories, which we named *tough_false_matches* and *easy_false_matches*, respectively, we need a simple method for understanding the difficulty of matching a pair. To that end, we created a simple Term Frequency - Inverse Domain Frequency (TF-IDF) model. First, we select all pairs of products having a *Jaccard similarity* score of at least *0.7*, then a new column is added to every product representing the TF-IDF encoding of that offer's title. Consequently, a *cosine similarity* score is computed between pairs that passed the *Jaccard similarity* filter. If the cosine similarity is larger than *0.66*, the TF-IDF model considers that pair a match. We selected the threshold for Jaccard Similarity by manually checking how many false negatives and false negatives are generated for various values. The purpose here is to obtain sufficiently large pools of each category to be able to create a large dataset in the next step.

The next step consists of comparing the results obtained by this model to the true labels known by matching offers on the *product_id* column. The tool of choice for looking at the comparison between the two sets of labels is a confusion matrix. Using it, true negatives become *easy_false_matches*, false positives become *tough_false_matches*. Additionally, we can divide the matching pairs into *easy_true_matches* (true positives) and *tough_true_matches* (false negatives).

We enforced the following quota per category to make sure that the dataset is not biased towards one of the categories: *5%* easy true matches, *5%* tough true matches, *30%* tough false matches, *40%* same brand false matches, *20%* easy false matches. It is worth noticing how the dataset contains far more negative examples than positives. Having an unbalanced dataset is a common trait of ER, and it represents the main reason why accuracy does not represent a good metric for ER tasks, as discussed in Section 3.1. The python code of the overall process is presented in Listing A.1. We included both tough and easy true, respectively false matches, because we want the model to be able to accurately predict all kinds of pairs. For example, if we only include tough pairs, there is a risk that the model will learn to focus on subtle differences every time, running the risk of mislabeling a lot of fairly obvious matches. We call this the `panprices_tvs` dataset.

3.3 Baseline model and data exploration

Having a baseline model to validate assumptions about the data and compare more complex models against is a common practice in the literature [8, 42]. For simplicity, we selected an already existing baseline model from previous research projects. The choice for the current project is to implement the Word co-occurrence model proposed by the authors of JointBERT [8]. This model works by splitting the text describing an offer into words, then looking at the words the two compared offers share in common, resulting in the co-occurrence matrix. Simple machine learning algorithms are then used to try to fit a prediction function to this matrix. The algorithms are: Bernoulli Naive Bayes, XGBoost, Random Forest (RF), Decision Tree (DT), Support vector machines (SVM), Logistic Regression. The full list of hyperparameters used for these experiments is listed in A.3. A randomized grid search over a defined hyperparameter space is performed for each of these, and the model best fitting the validation set is selected.

Model	**Dataset**	**F1**
Baseline	abt_buy	0.40
Baseline	amazon_google	0.44
Baseline	wdc_computers_medium	0.83
Baseline	panprices_tvs	0.76

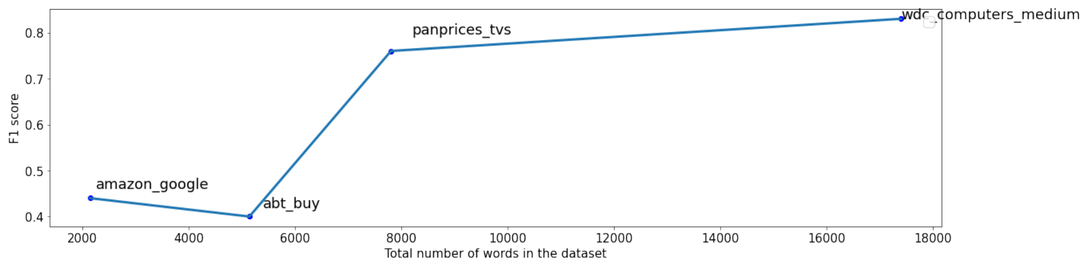
Table 3.3: F1 score of the baseline model on the studied datasets

Table 3.3 presents the results obtained by the baseline model on the studied datasets. One important takeaway from these results is the difference in performance between the first two open datasets (`abt_buy` and `amazon_google`) and the third one (`wdc_computers_medium`) is significantly high. One hypothesis to support this observation is that the number of words per offer in the first two datasets is lower than in the third one. Furthermore, there may be a more significant difference in vocabulary between the train and test splits for the first two datasets. For the rest of this section we will focus on testing these hypotheses.

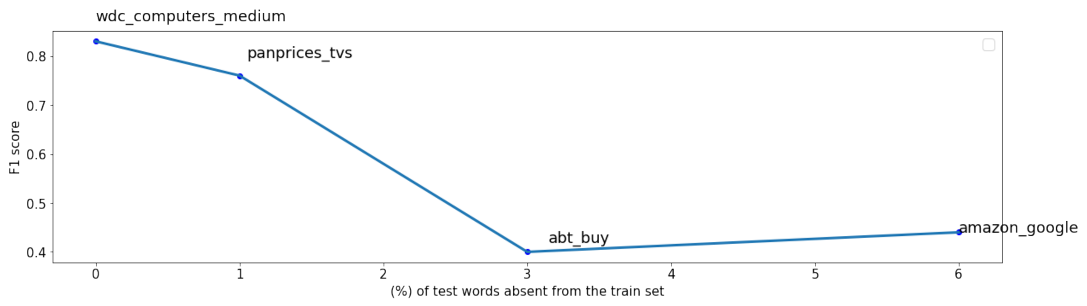
Further analysis of the data can be performed to explore these ideas. One could take a look at the global picture (Table 3.4), as well as grasp a more localized view (Table 3.5). Both validate the proposed hypotheses, shading light on the main difference between the studied datasets.

Dataset	**Total words**	**Train - Test word split (%)**			**Avg. words per offer**	
		Common	**Train only**	**Test only**	**Left offer**	**Right offer**
abt_buy	5,144	69%	28%	3%	46	15
amazon_google	2,146	53%	43%	6%	7	7
wdc_computers_medium	17,408	29%	71%	0%	78	73
panprices_tvs	7,810	50%	49%	1%	82	160

Table 3.4: Number of words between the train / test splits for the studied datasets



(a) Performance by total number of words



(b) Performance by percentage of words that are only present in the test set

Figure 3.1: Influence of the total number of words and the percentage of test only words on the performance (F1) of the baseline model

Two main things can be concluded by looking at Table 3.4: there are substantially more words present in `wdc_computers_medium` dataset compared to the other two, and this is as well the only dataset where all the words being used in the test set are present as well in the train set. In other words, in the case of the `wdc_computers_medium` dataset, the model has more information to base a decision on and it was trained to handle all possible tokens. The `panprices_tvs` dataset contains more words as well, and we can also see here that only a small percentage of the test words were not present in the training set. These findings explain the large difference in performances observed in Table 3.3, and are supporting the hypothesis we presented at the beginning of the section. Figure 3.1 offers a visual representation of our conclusions regarding the baseline model performance.

To understand the influence of the metrics above on the model's performance, a more local view is required. Table 3.5 shows a mislabeled example from the `abt_buy` dataset. These two offers represent a match, describing the same physical product, but the baseline model labels this pair as non-matching. OOV words are highlighted for interpretation.

Source	**Text**
A	panasonic kx-tga450b black 5.8 ghz cordless handset kxtga450b panasonic kx-tga450b black 5.8 ghz cordless handset kxtga450b frequency hopping digital spread spectrum technology answering system with compatible base unit call waiting caller id join in/privacy 2-way intercom voice scramble black finish
B	panasonic kx-tga450b cordless handset panasonic kx-tga450b additional handset for pankxtg4500b

Table 3.5: Mislabeled example from the abt_buy test set. Highlighted words are not present in the training set. Expected label: *match*, Baseline model label: *non-match*

An obvious issue here is that the model name *kx-tga450b* represents an OOV token. Because the model has never seen this token during training, it can not be considered when comparing the two texts. Practically the model is trying to predict without considering the OOV words at all. Such a task (upon eliminating words absent in the train set) becomes impossible even for a trained human because the other available words can not constitute a strong enough basis for making a decision. Therefore, the conclusion that OOV words are an issue for the baseline model stands and further explains the improved performance on the dataset where no such words are present (`wdc_computers_medium`).

3.4 Implementing the SoA

As discussed in Chapter 2, many solutions have been proposed for the ER problem. This project focuses on the sub-category of solutions based on TPLMs, the latest evolution of ER models, which currently represent the SoA.

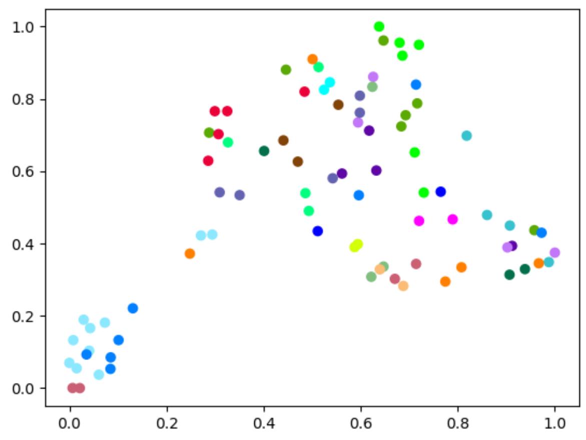
Specifically, the best results available at the moment of this writing are presented by the authors of R-SupCon [9]. Thus, this model represents the target of the SoA implementation.

Peeters *et al.,* [9] made their code available on GitHub. Their repository* serves as the basis for the implementation of the model. Nevertheless, to better understand the process the authors went through, we rewrote and adjusted the original code. The final source code can be viewed on a separate GitHub repository †.

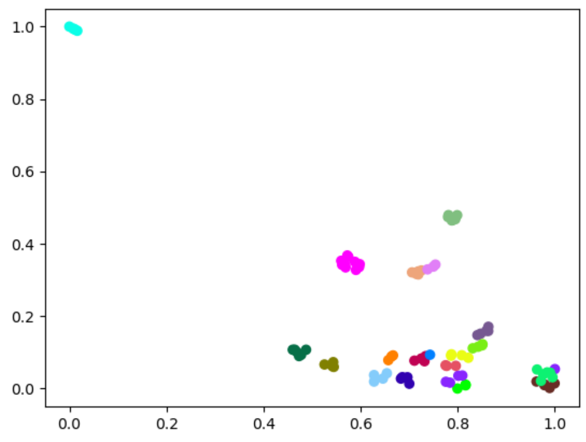
The main idea behind contrastive learning is to format the embedding space so that matching offers come close together. At the same time, offers presenting different products should be placed far in the embedding space. It originates in the field of Computer Vision (CV), where it was first devised as a self-supervised learning method [49]. Images would go through an augmenting process. The purpose of the algorithm was to bring together augmentations of the same image, learning how to group similar images in the embedding space. Later, the possibility of using labels with the contrastive loss function was introduced [46], instructing the model to group together images sharing the same label as well as augmentations from the same image.

* https://github.com/wbsg-uni-mannheim/contrastive-product-matching

† https://github.com/damianr13/master-thesis-kth



(a) Offer embeddings before contrastive pretraining



(b) Offer embeddings after contrastive pretraining

Figure 3.2: How the contrastive pretraining of R-SupCon affects offer embeddings of offers in the `wdc_computers_medium` dataset

In the case of ER, e-commerce offers need to be embedded instead of images. Figure 3.2 shows how the contrastive learning pretraining step modifies the embedding space. Each point in both charts represents an offer, and the same colour points represent matching offers. Only a sample of 20 products was selected to create these charts. While it may seem that a few groups are unnecessarily close to one another, it is worth pointing out that these charts are obtained after reducing the dimensions of the embeddings

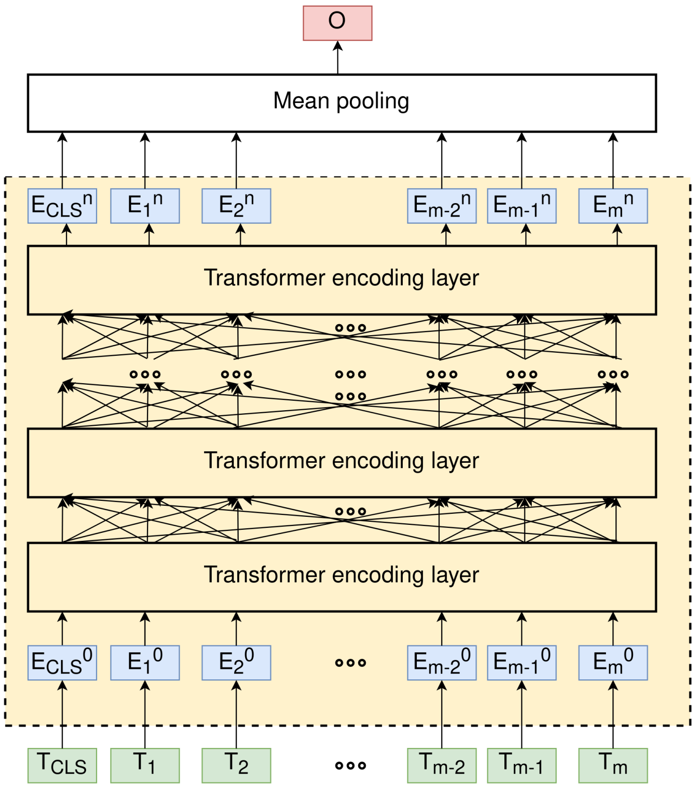


Figure 3.3: How the embedding of one offer is generated

from 312 to 2, using UMAP [*], and that a lot more products are present in the dataset. Even if some clusters of different clusters are close in this 2D space, we expect that other dimensions serve the purpose of distinguishing them in the higher order space. Moreover, some of the offers that were not selected in this sample will cover the white space visible in the upper left corner of the second sub-figure.

The architecture for obtaining the embedding of one offer is summarized in Figure 3.3. First the text describing an offer is tokenized and fed to the TPLM, enclosed in the yellow background in the figure. The TPLM produces a contextualized embedding for each of the fed tokens. In the next layer you take the average of the embeddings of all the tokens and we use that as the embedding of our offer. When a full batch of offers had undergone this process, we train the model with the contrastive objective to bring close matching offers in the embedding space.

[*] https://umap-learn.readthedocs.io/en/latest/

Items (images in the original works, offers in the case of R-SupCon and the current project) are individually embedded in the space. However, the final purpose of the model is to estimate the probability of two different items being similar. Thus, another step must be performed to obtain similarity results out of the model. One option is to calculate the distance (for instance, cosine or euclidean) between two embeddings and compare that to a threshold. Another option is to perform a second training stage, where a head is added on top of the siamese architecture, and the whole model is trained using the cross-entropy loss function. Authors of the original SupCon paper [46], and subsequently, the authors of R-SupCon [9] opted for the latter.

Figure 3.4 shows the complete training architecture with the classification head on top of the siamese neural network composed of the contrastive pre-trained model. Surrounded by the yellow background we have the TPLM that was pre-trained in the previous step. Similarly to how we did for the pre-training model, we obtain the embedding of an offer by averaging the embeddings of the tokens that make it up. In the final step, we concatenate the two embeddings, with their difference and their element-wise product, to form the feature vector upon which we base the prediction of whether the pair should be considered a match or not.

Having built the intuition behind the contrastive loss function, here is the formula devised by the authors of the original SupCon paper [46]:

$$\mathcal{L}_{sup} = \sum_{i \in I} \mathcal{L}_{sup}^{i} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_p / \tau\right)}{\sum_{a \in A(i)} \exp\left(\boldsymbol{z}_i \cdot \boldsymbol{z}_a / \tau\right)}$$

The first thing to remember is that the formula uses all the offers in one batch. The loss is calculated by summing up the losses obtained by considering each offer in the batch an anchor at a time (i.e. $i \in I$). For one anchor, the loss is obtained by summing up the losses between it and all the other items belonging to the same cluster (i.e. $p \in P(i)$). Cluster here means a group of offers referring to the same products, as indicated by the labels. Furthermore, the loss between an anchor item and a member of its cluster is calculated as the log of the ratio of the exponential distance between the two embeddings, and the sum of the exponential distance between the anchor and all the embeddings outside the cluster (i.e. $a \in A(i)$). The loss is scaled by the anchor's cluster's

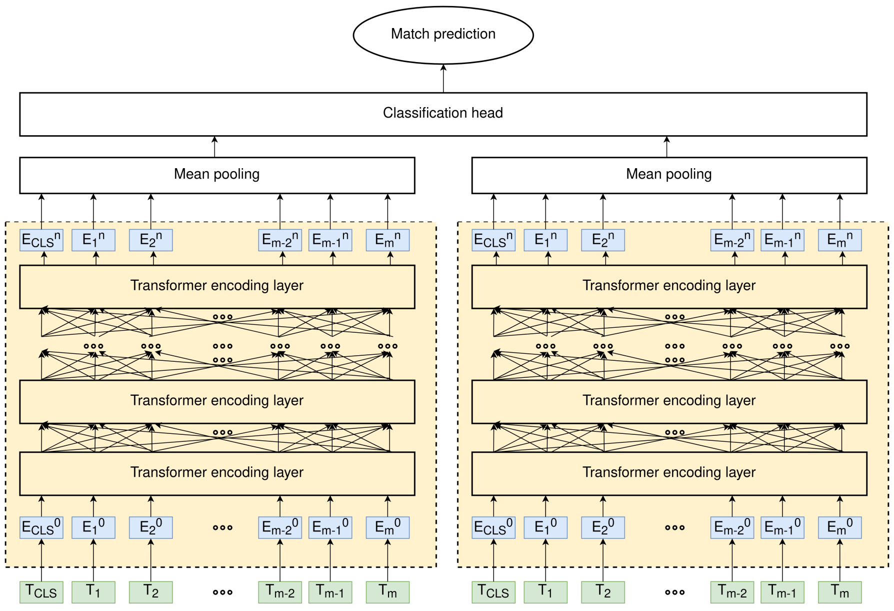

Match prediction
Classification head
Mean pooling
Mean pooling
Transformer encoding layer
E_{CLS}^n E_1^n E_2^n E_{m-2}^n E_{m-1}^n E_m^n
E_{CLS}^0 E_1^0 E_2^0 E_{m-2}^0 E_{m-1}^0 E_m^0
T_{CLS} T_1 T_2 T_{m-2} T_{m-1} T_m

Figure 3.4: Siamese architecture with the classifier head on top

size. Since the ratio inside the log function is always sub-unitary, the resulting loss would be negative. The prevent that, the result is multiplied by -1.

As previously stated, for the model to distinguish between two given offers, an additional training step must be performed. In this step, two offers are passed in parallel to the transformer as show in Figure 3.4. The binary cross entropy loss function is used for training the model to distinguish a tuple of offers:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

3.5 Introducing Spring R-SupCon

In the following subsection we focus on what we consider a weakness in R-SupCon, and propose a way to mitigate it. Next, we present our solution in the form of *Spring R-SupCon*. Finally, we provide a graphical intuition into the problem with R-SupCon, and why our fix addresses it.

3.5.1 A closer look at R-SupCon

The authors of R-SupCon [9] propose two ways of training the siamese model with the classifier head on top. The first option they test is to freeze the TPLM obtained after the pretraining step and use the binary cross-entropy loss to only train the classifier head that was added on top of the siamese network. The second idea they experimented with consists of propagating the binary cross-entropy loss in the whole network and training the weights in the BERT model alongside the classification head in the second step. The former is introduced as the *frozen* version, whilst the latter is presented as the *unfrozen* version of the model.

Empirically, they prove that the *frozen* version vastly outperforms the *unfrozen* one on all the studied datasets. The superior performance of the *frozen* model is to be expected because of the random initialization of the head that was added on top of the model. The contrastive learning step had the purpose of clustering together in the embedding space offers referring to the same products. Since the head's weights are randomly initialized, there is a high probability that offers belonging to the same cluster may be labelled as non-match or vice-versa. Thus, the randomly initialized head added on top of the siamese network results in high losses at the beginning of the training phase. If these high losses are then propagated into BERT, the property of grouping together similar offers in the embedding space is affected. In other words, the embedding space gets scrambled by the randomness present in the head, undoing the benefits of contrastive learning.

The two operational modes used introduced here as *frozen* and *unfrozen*, are widely known as *feature extracting* and *fine tuning*, respectively [50]. A TPLM is said to be used as a feature extractor when its internal weights are not trained anymore, and the model is only used for embedding a sentence (or sequence of words). On the other hand, fine-tuning a model to a specific task means further training the weights inside a model to better fit the final task it is meant to perform. Generally, given enough resources, we expect *fine tuning* to yield better results than *feature extracting*, which is not the case for R-SupCon. In this case, the first training step is fine-tuning the model, while the second step uses it only as a feature extractor.

3.5.2 From R-SupCon to Spring R-SupCon

The research question this project is trying to answer is then: "Are there any benefits to fine-tuning the TPLM with the binary cross-entropy loss function after the contrastive training step?". The first sub-question that arises is how to fine-tune the model using the binary cross-entropy loss function. Peeters *et al.,* [9] empirically show that the unfrozen model is performing poorly, so we discard this option.

For using the benefits of fine-tuning using the second loss function, this project proposes adding a third training step after the *frozen* training performed by the authors of R-SupCon [9]. This third training step is not a new one, it is the *unfrozen* step. The innovation here consists in performing both training modes one after the other instead of one or the other. The intuition dictates that since the head will have been fine-tuned by the *frozen* training step, the losses will be vastly lower than the ones obtained by directly performing the *unfrozen* step. Thus, the risk of scrambling the embeddings space diminishes drastically while allowing the TPLM to further adapt to the task of estimating the probability of two given offers to represent a match. We call our model *Spring R-SupCon*.

3.5.3 The effects of freezing the TPLM

Figure 3.2 shows how contrastive pre-training of the TPLM clusters together matching offers. As we explain in Section 3.4, the contrastive pretraining has to be followed by another phase of tuple classification to train the model. The Binary Cross Entropy (BCE) loss used for teaching the model how to distinguish between two offers can be propagated to the TPLM. If the loss is not propagated to the TPLM and only the head is trained with the tuple classification objective, the offer embeddings stay the same as in the second sub-figure (after contrastive pretraining).

On the other hand, if the BCE loss is propagated to the model as well, the TPLM's weight will change, resulting in significantly different embeddings. Figure 3.5 shows how the embeddings are affected if the model is directly trained with the tuple classification objective, starting with the randomly initialized head. The advantages of the contrastive learning phase have been lost, and the offers are no longer clustered according to a matching relation.

In this project we propose a new way of training the model, in which the

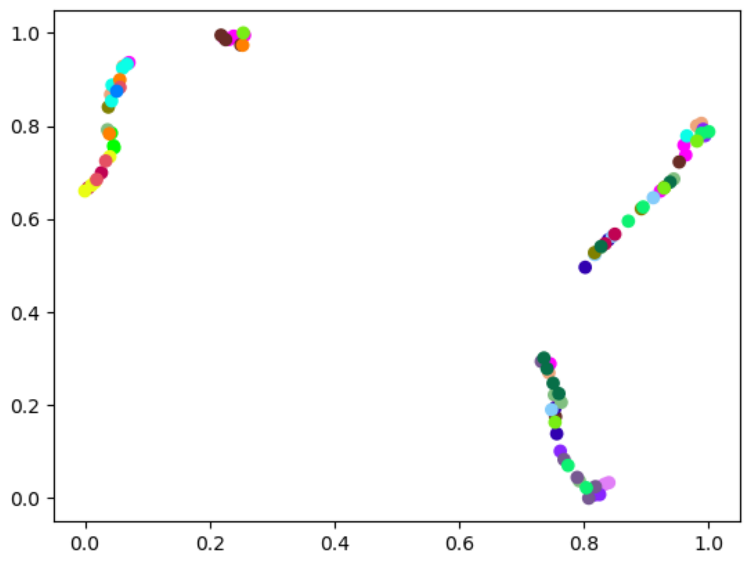


Figure 3.5: Resulting embeddings after training R-SupCon in *unfrozen* mode on the `wdc_computers_medium` dataset

TPLM is initially frozen to allow the classification head first to be pre-trained for the Tuple Classification task. This phase is then followed by a third training phase, in which the loss propagates through the entire model, allowing the TPLM to be fine-tuned to the Tuple Classification task as well. Figure 3.6 shows that not much changed in the embeddings created by the TPLM. This behaviour was expected since the head is not randomly initialized as was the case when performing the *unfrozen* experiment, but fine-tuned to the task.

3.6 Performance with low data volumes

For one of the use cases the host company has, it is not possible to use GTINs for creating a large dataset in a distant supervised fashion as proposed by previous research [48]. Instead, datasets must be manually labelled, which is costly and time-consuming. Therefore, it is interesting to understand how various models can perform under conditions of low data volumes.

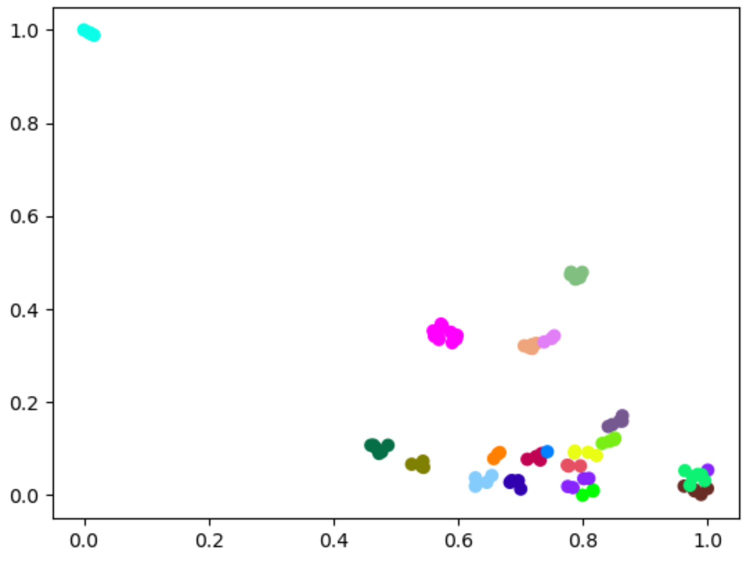


Figure 3.6: Resulting embeddings after training Spring R-SupCon in *frozen* and *unfrozen* mode subsequently on the `wdc_computers_medium` dataset

In this project we propose a method of artificially creating lower volume datasets from larger ones to simulate such environments. Although ER datasets consist of pairs of offers, it would not suffice to sample those. In a real-world environment, data arriving at an entity does not consist directly of pairs of offers, but the offers themselves are stored in a retailer's database. The pairs are then created using a blocking algorithm as explained in Chapter 2.

Let us take a concrete example where sampling pairs would create an inconsistent dataset. Similarly to the problem introduced in Chapter 2, we consider two data sources A, B. Let $m, n \in A$ and $x, y \in B$, and let the pairs $(m, x, 1), (m, y, 0), (n, x, 0), (n, y, 1) \in A \times B \times \{0, 1\}$ be the labeled pairs present in the dataset. A random pair sample can select the following pairs: $(m, y, 0), (n, x, 0)$. In this case, even though offers m, n, x and y are all present in the dataset, the matching relationship has been lost. This effect has significant damage on the contrastive learning step because the algorithm will try to push apart offers m and x and n and y respectively, since there is no more a link between them in the dataset.

Offers should be sampled instead of pairs, and then all pairs composed of two sampled offers shall be kept. Considering the case above, let us say that the algorithm samples m, n and x to be kept in the lower volume dataset. In other words, pairs $(m, x, 1)$, $(n, x, 0)$ will be sampled, keeping the consistency of the data. In this case, we disregard $(m, y, 0)$ and $(n, y, 1)$ since they contain a non-selected offer, *i.e.,* an offer that would be unknown in the real-world scenario.

Furthermore, we need to preserve a ratio of positive/negative matches similar to the one observed in the full dataset. Thus, after randomly sampling $A^s \subset A$, a small sample $B^m \subset B$ is drawn out of B in such a way that $\forall b \in B^m, \exists a \in A^s$, such as $(a, b, 1) \in A \times B \times \{0, 1\}$. Then we complete the sample with random elements $B^r \subset B$, and the final sample from the second data source is computed as $B^s = B^m \cup B^r$.

Listing A.2 presents the python method used for performing the sampling process described above.

3.7 The question of the model size

The authors of R-SupCon [9] used RoBERTa as the basis for their model. As seen in Chapter 2, RoBERTa is a large TPLM that holds the SoA in a range of NLP tasks. Nevertheless, it comes with the downside of hosting many parameters and requiring powerful resources and long training times. In the same chapter, we took a look at significantly smaller TPLMs (ALBERT, TinyBERT) that strive to keep the performance of their larger relatives.

In this section we study the benefits of using a larger TPLM in the context of ER. For this purpose, we compare the original R-SupCon model based on RoBERTa against a clone that uses TinyBERT instead.

Table 3.6 shows that the results obtained using TinyBERT are almost on par with the ones obtained by using RoBERTa. The difference between the two TPLMs is insignificant for `wdc_computers_medium`. The results obtained for the `abt_buy` dataset are worse when using TinyBERT, while the results for `amazon_google` are better than the ones observed when using RoBERTa.

Table 3.7 shows the resource consumption levels for both RoBERTa and

Model	**Dataset**	**RoBERTa** *	**TinyBERT**
R-SupCon	amazon_google	79.28	84.87
R-SupCon	abt_buy	94.29	90.05
R-SupCon	wdc_computers_medium	98.50	97.83

Table 3.6: Comparison of F1 scores obtained by using RoBERTa and TinyBERT inside R-SupCon. Results with * taken from the R-SupCon paper [9]

Model	**Metric**	**RoBERTa**	**TinyBERT**
R-SupCon	Energy consumption (kWh)	5.11	0.66
R-SupCon	Time required (h)	36	6

Table 3.7: Comparison of resource consumption observed in training R-SupCon by using RoBERTa and TinyBERT as the base TPLM. Experiments ran on GCP, see Chapter 4 for a detailed hardware description

TinyBERT. Training the model starting from TinyBERT requires six times less time and consumes about eight times less energy than the RoBERTa version. Therefore, correlating the data in Tables 3.6 and 3.7, we took the decision to carry on further experiments using TinyBERT.

Chapter 4

Results and Analysis

In this chapter we present and analyze the results obtained after performing our experiments. We start by providing an overview of the hardware and software we used (Section 4.1), then we show how our model compares to the other SoA models when dealing with low data volumes for cases where GTIN is not available and we need to rely on manually created datasets (Section 4.2). Finally, we show how Panprices AB can use current solutions for the TV category, for which we automatically created a dataset in the previous chapter (Section 4.3).

4.1 Experimental setup

We carried out all of our experiments on cloud platforms provided by Google and Oracle. The VM of choice on Google Cloud Platform (GCP) is n1-highmem-8 with an NVIDIA Tesla K80 GPU, while on Oracle Cloud, a VM.GPU3.1 instance with an NVIDIA Tesla V100 GPU will be used.

The language of choice for implementing the models studied as part of this project is Python 3.9, using the PyTorch framework and other well-established libraries from the data science toolkit: NumPy, pandas, scikit-learn. Data engineering tasks will be carried out using PySpark.

4.2 Experiments with lower data volumes

One of the use cases discussed with the host company referred to applying ER solutions when lower volumes of data are available. At the time of this writing, the company did not yet have an example dataset, so we applied the sampling process described in Section 3.6 to reduce the volume of existing public datasets.

Based on the three open datasets: `abt_buy`, `amazon_google` and `wdc_computers_medium` nine smaller sized datasets have been generated corresponding to the following sampling ratios: *0.25*, *0.50* and *0.75* applied to each of the original datasets. To understand the effect of the training set size on the performance of the model, the *size* of a dataset has been operationalized in four ways:

1. Number of offers
2. Number of pairs
3. Number of products
4. Number of matching pairs

Note that all these metrics refer to the size of the training set. The size of the test set has been unaffected by the sampling technique presented in Section 3.6. Table 4.1 show the size of the sampled datasets measured by the four metrics mentioned above.

We used three models to fit the generated data. The first is Ditto [7], which represented the SoA in TPLM architectures for ER before the R-SupCon's [9] publication. The other two are based on the R-SupCon paper. The difference between them consists of the way the model was trained. The *frozen* model is the one presented in the original paper, where the TPLM's parameters are frozen during tuple classification training. The *Spring R-SupCon* model was trained in the way proposed by the current project in which the tuple classification training phase is further divided into two parts. During the first, the model is trained exactly as the *frozen* variant, while for the second, the model is trained end-to-end, propagating the loss back into the TPLM.

For the third training phase, we used a learning rate of *1e-6* for *50* epochs. The hyperparameters used were identified by performing a grid search over a user-defined space, using the `amazon_google` dataset.

Figure 4.1 shows the results obtained in the lower data volumes experiments. The size space can be divided into three categories: *very low*

*

Dataset	Sample ratio	# pairs	# offers	# products	# matching pairs
amazon_google	0.25	970	669	539	133
	0.50	2,495	1,302	991	300
	0.75	4,667	2,029	1,448	503
	1.00	6,874	2,679	1,835	699
abt_buy	0.25	696	448	347	97
	0.50	2,051	910	621	253
	0.75	3,789	1,361	840	432
	1.00	5,743	1,838	994	616
wdc_computers_medium	0.25	1,033	1,151	560	251
	0.50	2,568	2,147	694	603
	0.75	4,421	2,943	738	1,006
	1.00	6,475	3,498	745	1,410

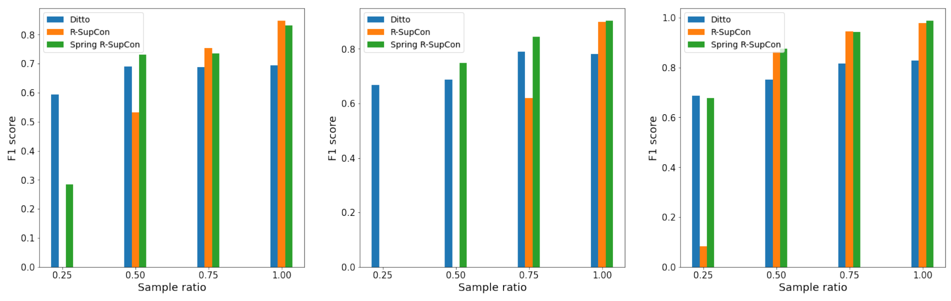
Table 4.1: Sizes of sampled datasets as shown by the four operationalizations

Model	**amazon_google**				**abt_buy**				**wdc_computers_medium**			
	0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00	0.25	0.50	0.75	1.00
Ditto	**59.31**	69.02	68.87	69.38	**66.66**	68.73	79.06	78.00	**68.62**	75.14	81.54	82.73
R-SupCon	0.00	53.22	**75.46**	**84.87**	0.00	0.00	62.04	**89.79**	8.30	86.33	**94.49**	**97.81**
Spring R-SupCon	28.37	**73.19**	73.60	**90.41**	0.00	**74.87**	**84.47**	83.23	67.75	**87.64**	**94.47**	**98.86**

Table 4.2: Comparison of the performance (F1) of three ER models on lower volume datasets

data volumes (left part of the charts) , *low data volumes* (middle part of the charts), *low to medium data volumes* (right part of the charts). Each of the tested models dominates its category. In other words, the ER solution of choice is not universal, and a decision should be taken according to the volume of the data one has available.

Ditto dominates the *very low volume* category, proving less “data-hungry” than contrastive based methods. This conclusion is expected because the contrastive pretraining phase needs many offers to map them in the embedding space. Otherwise, the value of using it significantly diminishes. Furthermore,



(a) amazon_google (b) abt_buy (c) wdc_computers_medium

Figure 4.1: Comparison of the performance (F1) of three ER models on lower volume datasets

because in Ditto, the offers are concatenated and given to the TPLM as one array of tokens, the model can learn distinguishing factors between two offers (inter-offer interactions) and use them more efficiently. On the other hand, R-SupCon uses a siamese network, which means an embedding is obtained for each offer before comparing them. Thus, R-SupCon cannot directly compare tokens belonging to the two offers, like a different model name or manufacturer. A particularly concerning result is the one obtained for the *0.25* sample of the `abt_buy` dataset. Both of the contrastive learning models obtain an F1 score of 0 for the given dataset. We explain this by the fact that the dataset does not contain sufficiently many offers of the same products, for the contrastive learning's clustering power to become useful. For such datasets, a case by case approach is more suitable as demonstrated by Ditto's performance.

As the amount of training data increases, the *Spring R-SupCon* model becomes better than Ditto. In the charts it can be seen how the *Spring R-SupCon* dominates both Ditto and *frozen* R-SupCon in the *low data volumes* category. These results can be explained by the fact that this training scheme benefits both from the contrastive pretraining and the ability to adapt to pairs of offers. Although our model still uses the siamese architecture and thus is not able to pick up slight differences as easy as Ditto, it still enabled the TPLM to learn differences in a pair through the third learning phase. This ability gives it a clear advantage over the *frozen* version of R-SupCon, and it bumps its performance above Ditto for the second category.

In the third category, the data volumes are high enough for the contrastive learning pretraining to matter much more. The model sees enough data during training time to learn a good enough separation of the embedding space. The classification head's task of splitting the embedding space into distinct areas becomes easier to achieve in this well-clustered space. Further training with the tuple classification objective in the third phase, where the TPLM is fine-tuned together with the head, is no longer beneficial. It is worth mentioning that even though the *Spring R-SupCon* model does not outperform the *frozen R-SupCon* model for this category, it does not affect the performance either. Thus, it is safe for a user to keep the third training phase because the results adding it are better or equal to the model that was not trained with it.

4.3 Panprices' data

As stated in Chapter 1, one of the main objectives of this project is to SoA solutions on data provided by Panprices AB to solve the company's issue with product matching. In Chapter 3, Subsection 3.2.2 describes the process of transforming the company's data into a suitable format for training ER models. This section focuses on analyzing the results obtained for the corresponding dataset.

Since this analysis makes the subject of an external report directed to the company, we included more metrics to understand the model's performance better. FPR and FNR were added because this is how the company's requirements were initially specified. Additionally, precision and recall were added to convey more details. Table 4.3 show the final results.

Model	Dataset	F1	FPR	FNR	Precision	Recall
word-cooc	panprices_tvs	75.86	5.51	10.81	66.00	89.18
Ditto	panprices_tvs	93.24	0.81	**6.75**	93.24	**93.24**
R-SupCon	panprices_tvs	**95.83**	**0.16**	**6.75**	**98.57**	**93.24**
Spring R-SupCon	panprices_tvs	**95.83**	**0.16**	**6.75**	**98.57**	**93.24**

Table 4.3: Proprietary dataset results

Based on the precision and FPR, it is clear that the model is aggressively labelling pairs as non-matching. The initial requirement received from the company was to achieve a FPR lower than 1%, and the model fulfils it with a good margin. The second requirement was that the FNR should be lower than

5%. This latter requirement is not satisfied by the current model, so adjusting the performance of this metric can constitute the basis for further R&D within the company.

Chapter 5

Conclusions

This final chapter acts as a retrospective of the work done (Sections 5.1 and 5.3), as well as a look forward into the future to what other ideas may be explored to improve the performance of ER systems (Section 5.2).

5.1 Summary

This project started with three main objectives: understanding the existing solutions for ER, implementing and improving the SoA and applying the gained knowledge to the real-world data provided by the host company. This section looks at each of them and assesses how they were covered.

We began by understanding the environment around existing ER solutions. Our work details how TPLMs works and provides a comprehensive list of SoA solutions available in the literature. More than presenting the different solutions, we also analysed the trend of technologies being used and compared key elements like main algorithms and training objectives for the selected models.

Moving on, we covered R-SupCon's [9] overall architecture and training method, with the intent of further understanding what makes it the SoA in e-commerce ER. We used our understanding to propose the idea behind Spring R-SupCon and set up a few experiments generated by the need of the host company to apply ER algorithms in low data volume environments. Our results show when the proposed model yields better results than the original method of training R-SupCon. It is also shown that it does not affect cases where the performance was not improved either.

Finally, we covered the third objective of applying the gained knowledge to the problems faced in the real world by the host company. Partly, the need for performing ER tasks in low data volumes environments is covered as described above. Even though no data from the company was available at the moment of this writing, the results obtained on public data constitute a solid base for the company to apply the knowledge in the future. Moreover, for the second use case, where GTINs are available, we described a method for creating a ER dataset. We split the generated dataset into a training set, validation set and a test set respectively and used it to train the proposed model, yielding promising results.

We restate the contribution of our work:

- we present *Spring R-SupCon*, an evolution of R-SupCon that is able to beat R-SupCon's performance on low data volume datasets by up to 74.47% F1 score
- Our experiments show that smaller language models like TinyBERT can perform almost as good as big language models at a fraction of the training time.
- we introduce a way to produce lower volume datasets from open datasets to diversify the conditions under which Entity Resolution (ER) models are tested

5.2 Future work

A possible way of improving the performance of Spring R-Supcon would be to include offer image data along the textual information. Previous works [51, 52] have already studied to possibility of using both visual and textual data for the task of ER. In the context of contrastive learning, we think it would be useful to separately embed the textual and visual representations of offers and concatenate them to obtain a higher order dimensional space, thus allowing the model to use more information when separating distinct offers. A challenge when it comes to this task is the lack of open datasets that include both visual and textual information. Wilke *et al.,* [52] overcome this by modifying the process through which Primpeli *et al.,* [47] created the Web Data Commons

(WDC) dataset, to acquire product images as well as textual data.

A recent article by another group of researchers at Amazon [53] proposed Trans-Encoder, a learning scheme in which a bi-encoder and a cross-encoder can learn from one another through distillation. A bi-encoder is a model that individually encodes every item available (offer in the case of this project) and then compares the two embeddings. On the other hand, a cross-encoder encodes a pair of items (a pair of offers) benefiting from inter-sentence interactions. From the work of the current project, R-SupCon is a bi-encoder, while Ditto is a cross-encoder. Therefore, a future project in ER could have a look at the possibility of making the two architectures interact through the mutual distillation learning scheme.

As mentioned in Chapter 1 some things like the blocking algorithm are intentionally left out of the current project due to time limitations. Nevertheless, blocking is an essential part of the ER pipeline and requires its investigation. Thus, blocking can represent a good target for a future related project.

In Chapter 2, we shortly described DIAL [44] among other possible solutions. One of the most important of DIAL's particularity is that it describes an *active learning* pipeline. In other words, it focuses on the ability of the model to learn actively by recommending potential matches to a human labeller and then performing a training phase using the newly acquired data. This method of using ER models can prove useful when relying on humans to manually label pairs of offers, as is the second use case of the host company. DIAL's matching algorithm is similar to the one proposed in Ditto [7]. Naturally, one possible future project can focus on including a contrastive learning approach, similar to the one introduced by R-SupCon [9], in an *active learning* ER pipeline.

5.3 Reflections

We have demonstrated the ability of using contrastive learning in the field of Entity Resolution (ER) for low volume datasets with an improvement of up to 12% F1 score compared to the previous SoA. Our work shows that matching methods based on small TPLMs have the ability to obtain good results even in contexts where ER datasets are not easy to create. We further proposed a method for sampling existing datasets, to create lower volume instances of them and we showed our process for generating a dataset for ER, when data for distant supervised learning is available.

References

[1] N. Barlaug and J. A. Gulla, "Neural networks for entity matching: A survey," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 3, pp. 1–37, 2021. [Pages 2 and 8.]

[2] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean answers over dirty databases: A probabilistic approach," in *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 2006, pp. 30–30. [Page 4.]

[3] F. Cusmai, C. Aiello, M. Scannapieco, and T. Catarci, "Record linkage as a multiobjective optimization problem solved by evolutionary computation," in *Proceedings of the 1st International Workshop on Emergent Semantics and cooperaTion in opEn systEMs*. Citeseer, 2008. [Page 4.]

[4] 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," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1581–1584, 2016. [Pages 4, 9, 16, 17, 18, and 24.]

[5] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and 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. [Pages 4, 9, 17, 18, 19, 20, and 24.]

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages 4, 10, 11, 13, and 14.]

[7] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, “Deep entity matching with pre-trained language models,” *arXiv preprint arXiv:2004.00584*, 2020. [Pages 4, 7, 9, 20, 21, 24, 25, 27, 45, and 52.]

[8] R. Peeters and C. Bizer, “Dual-objective fine-tuning of bert for entity matching,” *Proceedings of the VLDB Endowment*, vol. 14, no. 10, pp. 1913–1921, 2021. [Pages 4, 7, 9, 21, 24, and 30.]

[9] ——, “Supervised contrastive learning for product matching,” *arXiv preprint arXiv:2202.02098*, 2022. [Pages viii, 4, 22, 24, 25, 27, 33, 36, 38, 39, 42, 43, 45, 50, and 52.]

[10] U. Brunner and K. Stockinger, “Entity matching with transformer architectures-a step forward in data integration,” in *International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*. OpenProceedings, 2020. [Pages 4, 7, and 20.]

[11] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, “Comparative analysis of approximate blocking techniques for entity resolution,” *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 684–695, 2016. [Page 6.]

[12] G. Papadakis, D. Skoutas, E. Thanos, and T. Palpanas, “Blocking and filtering techniques for entity resolution: A survey,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–42, 2020. [Page 6.]

[13] W. Zhang, H. Wei, B. Sisman, X. L. Dong, C. Faloutsos, and D. Page, “Autoblock: A hands-off blocking framework for entity matching,” in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 744–752. [Page 6.]

[14] A. Doan, A. Halevy, and Z. Ives, *Principles of data integration*. Elsevier, 2012. [Page 6.]

[15] K. Nozaki, T. Hochin, and H. Nomiya, “Semantic schema matching for string attribute with word vectors,” in *2019 6th International Conference on Computational Science/Intelligence and Applied Informatics (CSII)*. IEEE, 2019, pp. 25–30. [Page 6.]

[16] N. Valstar, F. Frasincar, and G. Brauwers, “Apfa: Automated product feature alignment for duplicate detection,” *Expert Systems with Applications*, vol. 174, p. 114759, 2021. [Page 6.]

[17] I. P. Fellegi and A. B. Sunter, “A theory for record linkage,” *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969. [Pages 8 and 16.]

[18] J. Pennington, R. Socher, and 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. [Pages 10 and 18.]

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013. [Page 10.]

[20] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, “Advances in pre-training distributed word representations,” in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018. [Pages 10 and 18.]

[21] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, “What does bert look at? an analysis of bert’s attention,” *arXiv preprint arXiv:1906.04341*, 2019. [Page 13.]

[22] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” *arxiv:1802.05365v2*, 02 2018. [Page 14.]

[23] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018. [Page 14.]

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. [Page 14.]

[25] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020. [Page 14.]

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018. [Pages 14, 15, and 20.]

[27] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019. [Pages 14 and 15.]

[28] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019. [Page 15.]

[29] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019. [Pages 15 and 16.]

[30] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019. [Page 15.]

[31] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019. [Pages 15 and 16.]

[32] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New phytologist*, vol. 11, no. 2, pp. 37–50, 1912. [Page 16.]

[33] V. Levenshtein, "Leveinshtein distance," 1965. [Page 16.]

[34] A. Singhal *et al.*, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001. [Page 16.]

[35] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. [Page 17.]

[36] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1454–1467, 2018. [Pages 18 and 24.]

[37] C. Zhao and Y. He, "Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning," in *The World Wide Web Conference*, 2019, pp. 2413–2424. [Pages 18 and 24.]

[38] J. Li, Z. Dou, Y. Zhu, X. Zuo, and J.-R. Wen, "Deep cross-platform product matching in e-commerce," *Information Retrieval Journal*, vol. 23, no. 2, pp. 136–158, 2020. [Pages 19 and 24.]

[39] Z. Wang, B. Sisman, H. Wei, X. L. Dong, and S. Ji, "Cordel: A contrastive deep learning approach for entity linkage," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1322–1327. [Pages 19, 21, and 24.]

[40] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6. Morgan-Kaufmann, 1993. [Online]. Available: https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf [Page 19.]

[41] R. Peeters, C. Bizer, and G. Glavaš, "Intermediate training of bert for product matching," *small*, vol. 745, no. 722, pp. 2–112, 2020. [Pages 20 and 24.]

[42] J. Tracz, P. I. Wójcik, K. Jasinska-Kobus, R. Belluzzo, R. Mroczkowski, and I. Gawlik, "Bert-based similarity learning for product matching," in *Proceedings of Workshop on Natural Language Processing in E-Commerce*, 2020, pp. 66–75. [Pages 20, 24, and 30.]

[43] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019. [Page 20.]

[44] A. Jain, S. Sarawagi, and P. Sen, "Deep indexed active learning for matching heterogeneous entity representations," *arXiv preprint arXiv:2104.03986*, 2021. [Pages 21, 24, and 52.]

[45] M. Trabelsi, J. Heflin, and J. Cao, "Dame: Domain adaptation for matching entities," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, 2022, pp. 1016–1024. [Pages 21 and 24.]

[46] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 661–18 673, 2020. [Pages 22, 33, and 36.]

[47] A. Primpeli, R. Peeters, and C. Bizer, "The wdc training dataset and gold standard for large-scale product matching," in *Companion Proceedings of The 2019 World Wide Web Conference*, 2019, pp. 381–386. [Pages 22 and 51.]

[48] R. Peeters, A. Primpeli, B. Wichtlhuber, and C. Bizer, "Using schema.org annotations for training and maintaining product matchers," in *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, 2020, pp. 195–204. [Pages 22, 27, 28, and 40.]

[49] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607. [Page 33.]

[50] L. Tunstall, L. von Werra, and T. Wolf, *NATURAL LANGUAGE PROCESSING WITH TRANSFORMERS : building language applications with hugging face*. O'reilly Media, 2022. [Page 38.]

[51] D. Zhang, Z. Li, X. Wang, K.-L. Tan, and G. Chen, "Towards one-size-fits-many: Multi-context attention network for diversity of entity resolution tasks," *IEEE Transactions on Knowledge and Data Engineering*, 2021. [Page 51.]

[52] M. Wilke and E. Rahm, "Towards multi-modal entity resolution for product matching." in *GvDB*, 2021. [Page 51.]

[53] F. Liu, Y. Jiao, J. Massiah, E. Yilmaz, and S. Havrylov, "Trans-encoder: Unsupervised sentence-pair modelling through self-and mutual-distillations," *arXiv preprint arXiv:2109.13059*, 2021. [Page 52.]

Appendix A

Code fragments

Listing A.1: Code snippet showing the algorithm for generating a proprietary dataset

```
tfidf_results = perform_tfidf_analysis(offers_df)
true_matches = generate_true_pairs(offers_df)

# extract the confusion matrix of the TF-IDF model
categories_dict =
    obtain_intermediate_categories(tfidf_results,
    true_matches)

same_brand_false_matches =
    generate_same_brand_false_matches_from_dataframe(offers_df)
random_false_matches =
    generate_random_false_matches_from_dataframe(offers_df)

# remove duplicates from same brand false matches
    category
already_known_matches = functools.reduce(lambda a, b:
    a.union(b), categories_dict.values())
same_brand_false_matches =
    same_brand_false_matches.subtract(already_known_matches)

# remove duplicates from easy false matches category
    (easy_false_matches)
already_known_matches =
    already_known_matches.union(same_brand_false_matches)
```

```
random_false_matches =
    random_false_matches.subtract(already_known_matches)

# add the new two categories to the dictionary
    containing the data for all different categories
categories_dict[RANDOM_FALSE_MATCHES_KEY] =
    random_false_matches
categories_dict[SAME_BRAND_FALSE_MATCHES_KEY] =
    same_brand_false_matches

# drop duplicates
categories_dict = {k: drop_duplicates_textual(v) for k,
    v in categories_dict.items()}

# perform sampling and train, test, validation split
train_set, test_set, validate_set =
    compose(categories_dict)
```

Listing A.2: Dataset sampling method applied on a pandas DataFrame

```
def _sample_data(self, df: DataFrame) -> DataFrame:
    # take a sample of left ids
    left_ids = pd.Series(df['left_id'].unique()).sample(
            frac=self.config.train_sample_frac)

    # apply weights to second datasource offers
    # to increase the probability of sampling offers
    # involved in many pairs (matching or non-matching)
    weighted_df = df[df['left_id'].isin(left_ids)].copy()
    weights = weighted_df[['left_id',
        'right_id']].groupby('right_id').count().rename(
        columns={'left_id': 'right_weight'})
    weighted_df = weighted_df.merge(weights,
        left_on='right_id', right_index=True)

    # for a sampled id in the first datasource sample one
        id
    # on the right that id is paired with to ensure that
        the
    # sampled ids are indeed kept
    right_paired_ids = weighted_df.groupby('left_id',
        group_keys=False).apply(
        lambda s: s.sample(1,
           weights=weighted_df['right_weight'])
        )['right_id'].unique()
    right_paired_ids = pd.Series(right_paired_ids)

    # sample matching ids to ensure preservation of
    # negative / positive ratio
    right_matching_ids = df[df['left_id'].isin(left_ids)
        & (df['label'] == 1)]['right_id'] \
        .unique()
    right_matching_ids = pd.Series(right_matching_ids)

    # complete with randomly sampled ids on the right
    right_ids = right_matching_ids.sample(
           frac=self.config.train_sample_frac)
    right_ids = pd.concat([right_ids,
        right_paired_ids]).unique()
    right_ids = pd.Series(right_ids)

    right_all_ids = pd.Series(df['right_id'].unique())

    already_sampled_frac = len(right_ids) /
        len(right_all_ids)
    right_ids_pool = df[(df['left_id'].isin(left_ids))
                    & (~df['right_id'].isin(right_ids))]
                    ['right_id'].unique()
    right_ids_pool = pd.Series(right_ids_pool)

    to_sample = (self.config.train_sample_frac -
        already_sampled_frac) * len(right_all_ids) /
        len(right_ids_pool)
    right_ids = pd.concat([right_ids,
        right_ids_pool.sample(frac=to_sample)])

    # select pairs where both ids have been sampled
    return df[df['left_id'].isin(left_ids) &
        df['right_id'].isin(right_ids)]
```

Listing A.3: Hyperparameter space for the Word Co-occurence model

```
{
 "classifiers": [
    {
      "name": "bernoulli",
      "params": {}
    },
    {
      "name": "xgboost",
      "params": {
        "learning_rate": [0.1, 0.01, 0.001],
        "gamma": [0.01, 0.1, 0.3, 0.5, 1, 1.5, 2],
        "max_depth": [2, 4, 7, 10],
        "colsample_bytree": [0.3, 0.6, 0.8, 1],
        "subsample": [0.2, 0.4, 0.5, 0.6, 0.7],
        "reg_alpha": [0, 0.5, 1],
        "reg_lambda": [1, 1.5, 2, 3, 4.5],
        "min_child_weight": [1, 3, 5, 7],
        "n_estimators": [100],
        "n_jobs": [4]
      }
    },
    {
      "name": "random_forest",
      "params": {
        "n_estimators": [
          100
        ],
        "max_features": [
          "sqrt",
          "log2",
          null
        ],
        "max_depth": [
          2,
          4,
          5,
          10
        ],
        "min_samples_split": [
          2,
          5,
          10,
          20
        ],
        "min_samples_leaf": [
          1,
          2,
          4,
          8
        ],
        "class_weight": [
          null,
          "balanced_subsample"
        ],
        "n_jobs": [
          4
        ]
      }
    },
    {
      "name": "decision_tree",
      "params": {
        "max_features": ["sqrt", "log2", null],
        "max_depth": [2, 4, 7, 10],
        "min_samples_split": [2, 5, 10, 20],
        "min_samples_leaf": [1, 2, 4, 8],
        "class_weight": [null, "balanced"]
      }
    },
    {
      "name": "linear_svc",
      "params": {
        "C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
        "class_weight": [null, "balanced"],
        "dual": [false],
        "max_iter": [10000]
      }
    },
    {
      "name": "logistic_regression",
      "params": {
        "C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
        "class_weight": [null, "balanced"],
        "solver": ["liblinear"]
      }
    }
```

```
  ]
}
```

Appendix B

Managing infrastructure with Terraform

Using a cloud platform such as Oracle Cloud Platform comes with the advantage that we only pay for what resources we are using, without needing to buy the required hardware upfront. For research projects this brings a lot of flexibility and helps us better manage costs.

One issue we encountered while working on the current project is that Oracle keeps their GPUs occupied if they are attached to an instance even if that instance is shut down. In other words, even if no experiments are being performed, while the instance still exists we are being billed for the time the GPU stays attached.

Naturally, the solution is to destroy the instance on which we run the experiments after each use instead of shutting it down. We want to be able to do that, without having to reconfigure the VM every time we launch new experiments. To do so, we can keep the boot storage of the machine, and use that when creating a new instance to continue working from where we left off. Another good practice is to keep a separate block volume where we keep the data and code for the project, such as we don't need to move everything in case we want to run the experiments on a VM with a different shape, potentially incompatible with the boot volume.

Having to manually recreate the instance every time still represents an annoying overhead to getting started with our work. This is where Terraform [*]

[*] `https://www.terraform.io/`

comes in handy. It is a tool for managing infrastructure as code across multiple suppliers including Oracle, Google or Amazon.

To automate the deployment of infrastructure we created the script presented in Listing B.1. Some information has been hidden and is marked by surrounding it with < > for example the id of the used boot volume is represented here as <*Boot_Volume_Id*>. Essential here are the *source_details* where we specify that we need the instance to be base on the existing boot volume and the option *preserve_boot_volume* that ensures the boot volume will outlive the instance.

Listing B.1: Terraform script for launching an instance

```
provider "oci" {}

resource "oci_core_instance" "workspace_instance" {
  agent_config {
    ...
  }
  availability_config {
    recovery_action = "RESTORE_INSTANCE"
  }
  availability_domain = <AD>
  compartment_id = <Compartment_Id>
  create_vnic_details {
    assign_private_dns_record = "true"
    assign_public_ip     = "true"
    subnet_id            = <Subnet_Id>
  }
  display_name = "MasterThesisWorkspace"
  instance_options {
    are_legacy_imds_endpoints_disabled = "false"
  }
  metadata = {
    "ssh_authorized_keys" = <SSH_Key>
  }
  shape = "VM.GPU3.1"
  source_details {
    source_id = <Boot_Volume_Id>
    source_type = "bootVolume"
  }
  preserve_boot_volume = true
}

resource "oci_core_volume_attachment"
    "master_storage_attachement" {
  attachment_type = "iscsi"
  instance_id = oci_core_instance.workspace_instance.id
  volume_id    = <Block_Volume_Id>
}
```

Moreover we would like the instance to automatically stop if no more work is required of it. This allows us to let it run experiments over night without being afraid that we might pay extra for hours when no actual work was performed. To do so, we can use a monitoring alarm that triggers a cloud function. The alarm would watch the CPU utilization of the instance as an indicator of whether the machine is being used or not. After an hour of inactivity it triggers a cloud function that calls the *terraform destroy* action on the stack created with the script above. This alarm can be defined as well using terraform as seen in Listing B.2.

Listing B.2: Alarm for destroying the instance when it is not being used

```
resource "oci_monitoring_alarm" "associated_alarm" {
  compartment_id    = <Compartment_Id>
  destinations      = [<Topic_Id>]
  display_name      = "MasterThesisWorkspaceWatchdog"
  is_enabled        = true
  metric_compartment_id = <Compartment_Id>
  namespace         = "oci_computeagent"
  query             = "CpuUtilization[1h]{resourceId =
      \"${oci_core_instance.workspace_instance.id}\"}.max()
      <= 5"
  severity          = "Critical"

  message_format = "PRETTY_JSON"
}
```

Appendix C

Reproducibility across VMs using docker

For the scope of this project, experiments were run at times on different hardware, across Oracle and Google Cloud. Even when different VMs, of different shapes, were used we still wanted to keep consistent result across different runs. The first step for achieving this is of course seeding all the random generators with the same number, but that was not enough. We needed a way to ensure that the environment in which experiments are run is the same every time. For this purpose, we encapsulated out project in a docker instance, with the help of the *Dockerfile* script presented in Listing C.1

Listing C.1: Dockerfile used for consistency across different environments

```
FROM nvidia/cuda:11.4.0-base-ubuntu20.04

ENV TZ=Europe/Stockholm
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime &&
   echo $TZ > /etc/timezone

RUN apt-get update && apt-get install -y python3.9
   python3-pip

RUN ln -s /usr/bin/python3.9 /usr/bin/python

# upgrade pip
RUN python -m pip -q install pip --upgrade

COPY ./requirements.txt
```

```
   /home/root/thesis/requirements.txt
WORKDIR /home/root/thesis
RUN ls
RUN python -m pip install -r requirements.txt

COPY . /home/root/thesis
RUN python setup.py install

ENV WANDB_LOG_MODEL=true

CMD python -m src.main --debug
```

For DIVA

{
”Author1”: { ”Last name”: ”Damian”,
”First name”: ”Robert-Andrei”,
”Local User Id”: ”19970113-T956”,
”E-mail”: ”radamian@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
”Cycle”: ”2”,
”Course code”: ”DA258X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ””
,”programcode”: ”CLNS”
,”Degree”: ”Master’s degree”
,”subjectArea”: ”Cloud and Network Infrastructure”
},
”Title”: {
”Main title”: ”Finding duplicate offers in the online marketplace catalogue using transformer based methods”,
”Subtitle”: ”An exploration of transformer based methods for the task of entity resolution”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”Hitta dubbletter av erbjudanden i online marknadsplatskatalog med hjälp av transformer-baserade metoder”,
”Subtitle”: ”En utforskning av transformer-baserad metoder för uppgiften att deduplicera”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Peña”,
”First name”: ”Francisco J.”,
”E-mail”: ”frape@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”Examiner1”: { ”Last name”: ”Matskin”,
”First name”: ”Mihhail”,
”E-mail”: ”misha@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”Cooperation”: { ”Partner_name”: ”Panprices AB”},
"National Subject Categories": "10201, 10206",
”Other information”: {”Year”: ”2022”, ”Number of pages”: ”xi,70”},
”Series”: { ”Title of series”: ”TRITA-EECS-EX” , ”No. in series”: ”2021:00” },
”Opponents”: { ”Name”: ”Luca Colasanti”},
”Presentation”: { ”Date”: ”2022-06-27 12:00”
,”Language”:”eng”
,”Room”: ”via Zoom https://kth-se.zoom.us/j/61618788736”
,”City”: ”Stockholm” },
”Number of lang instances”: ”3”,
”Abstract[eng]”: €€€€

The amount of data available on the web is constantly growing, and e-commerce websites are no exception. Considering the abundance of available information, finding offers for the same product in the catalogue of different retailers represents a challenge. This problem is an interesting one and addresses the needs of multiple actors. A customer is interested in finding the best deal for the product they want to buy. A retailer wants to keep up to date with the competition and adapt its pricing strategy accordingly. Various services already offer the possibility of finding duplicate products in catalogues of e-commerce retailers, but their solutions are based on matching a Global Trade Identification Number (GTIN). This strategy is limited because a GTIN may not be made publicly available by a competitor, may be different for the same product exported by the manufacturer to different markets or may not even exist for low-value products. The field of Entity Resolution (ER), a sub-branch of Natural Language Processing (NLP), focuses on solving the issue of matching duplicate database entries when a deterministic identifier is not available. We investigate various solutions from the the field and present a new model called \emph{Spring R-SupCon} that focuses on low volume datasets. Our work builds upon the recently introduced model, R-SupCon, introducing a new learning scheme that improves R-’SupCons performance by up to 74.47\% F1 score, and surpasses Ditto by up 12\% F1 score for low volume datasets. Moreover, our experiments show that smaller language models can be used for ER with minimal loss in performance. This has the potential to extend the adoption of Transformer-based solutions to companies and markets where datasets are difficult to create, like it is the case for the Swedish marketplace Fyndiq.

€€€€,
”Keywords[eng]”: €€€€
Transformers, Language Models, Deep Neural Networks, Entity Resolution, Duplicate Detection, Entity Matching, Record Linkage, Contrastive Learning, e-commerce €€€€,
”Abstract[swe]”: €€€€

Mängden data på internet växer konstant och e-handeln är inget undantag. Konsumenter har idag många valmöjligheter varifrån de väljer att göra sina inköp från. Detta gör att det blir svårare och svårare att hitta det bästa erbjudandet. Även för återförsäljare ökar svårigheten att veta vilken konkurrent som har lägst pris. Det finns tillgängliga lösningar på detta problem men de använder produktunika identifierare såsom Global Trade Identification Number (förkortat “”GTIN). Då det finns en rad utmaningar att bara förlita sig på lösningar som baseras på GTIN behövs ett alternativt tillvägagångssätt. GTIN är exempelvis inte en offentlig information och identifieraren kan dessutom vara en annan när samma produkt erbjuds på en annan marknad. Det här projektet undersöker alternativa lösningar som inte är baserade på en deterministisk identifierare. Detta projekt förlitar sig istället på text såsom produktens namn för att fastställa matchningar mellan olika erbjudanden. En rad olika implementeringar baserade på maskininlärning och djupinlärning studeras i detta projekt. Projektet har dock ett särskilt fokus på “”Transformer-baserade språkmodeller såsom BERT. Detta projekt visar hur man generera proprietär data. Projektet föreslår även ett nytt inlärningsschema och bevisar dess fördelar.

€€€€,
”Keywords[swe]”: €€€€
Transformers, Språkmodeller, Djupinlärning, Entitetserkännande, Dubblettdetektering, Entitetsmatchning, Rekordkoppling, e-handel €€€€,
”Abstract[fre]”: €€€€

Le volume des données qui se trouve sur l’internet est en une augmentation constante et les commerces électroniques ne font pas note discordante. Le consommateur a aujourd’hui beaucoup des options quand il decide d’où faire son achat. Trouver le meilleur prix devient de plus en plus difficile. Les entreprises qui gerent cettes plates-formes ont aussi la difficulté de savoir en tous moments lesquels de ses concurrents ont le meilleur prix. Il y-a déjà des solutions en ligne qui ont l’objectif de résoudre ce problème, mais ils utilisent un identifiant de produit unique qui s’appelle Global Trade identification number (ou GTIN). Plusieurs difficultés posent des barriers sur cette solution. Par exemple, GTIN n’est pas public peut-être, ou des GTINs différents peut-être assigne par la fabricante au même produit pour distinguer des marchés différents. Ce projet étudie des solutions alternatives qui ne sont pas basées sur avoir un identifiant unique. On discute des methods qui font la décision en fonction du nom des produits, en utilisant des algorithmes d’apprentissage automatique ou d’apprentissage en profondeur. Le projet se concentre sur des solutions avec ”Transformer” modèles de langages, comme BERT. On voit aussi comme peut-on créer un ensemble de données propriétaire pour enseigner le modèle. Finalement, une nouvelle method d’apprentissage est proposée et analysée.

€€€€,
”Keywords[fre]”: €€€€
Transformers, Modèles de langage, Apprentisage en profondeur, Résolution d’entité, Détection de doublons, Apprentisage contrastif, commerce électronique €€€€,
}