NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

MSc THESIS

AutoER: Auto-Configuring Entity Resolution pipelines

Konstantinos A. Nikoletos

Supervisor: **Manolis Koubarakis,** Professor

Co-Supervisors: **George Papadakis,** Associate Researcher
Vasilis Efthymiou, Professor
Konstantinos Stefanidis, Professor

ATHENS

SEPTEMBER 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

AutoER: Αυτόματη Παραμετροποίηση Entity Resolution μεθόδων

Κωνσταντίνος Α. Νικολέτος

Επιβλέπων: **Μανόλης Κουμπαράκης,** Καθηγητής

Συνεπιβλέποντες: **Γιώργος Παπαδάκης,** Συνεργαζόμενος Ερευνητής
Βασίλης Ευθυμίου, Συνεργαζόμενος Καθηγητής
Κωνσταντίνος Στεφανίδης, Συνεργαζόμενος Καθηγητής

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2024

MSc THESIS

AutoER: Auto-Configuring Entity Resolution pipelines

Konstantinos A. Nikoletos
S.N.: 7115112200022

SUPERVISOR: **Manolis Koubarakis,** Professor

COSUPERVISORS: **George Papadakis,** Associate Researcher
Vasilis Efthymiou, Professor
Konstantinos Stefanidis, Professor

THESIS COMMITTEE: **Dimitrios Gunopoulos,** Professor
Alexandros Ntoulas, Professor
Manolis Koubarakis, Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

AutoER: Αυτόματη Παραμετροποίηση Entity Resolution μεθόδων

Κωνσταντίνος Α. Νικολέτος
Α.Μ.: 7115112200022

ΕΠΙΒΛΕΠΩΝ: **Μανόλης Κουμπαράκης,** Καθηγητής

ΣΥΝΕΠΙΒΛΕΠΟΝΤΕΣ: **Γιώργος Παπαδάκης,** Συνεργαζόμενος Ερευνητής
Βασίλης Ευθυμίου, Συνεργαζόμενος Καθηγητής
Κωνσταντίνος Στεφανίδης, Συνεργαζόμενος Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Δημήτριος Γουνόπουλος,** Καθηγητής
Αλέξανδρος Ντούλας, Καθηγητής
Μανόλης Κουμπαράκης, Καθηγητής

ABSTRACT

The same real-world entity (e.g., a movie, a restaurant, a person) may be described in various ways on different datasets. Entity resolution (ER) is the problem of finding such descriptions that refer to the same entity, this way improving data quality and therefore, data value. However, an ER pipeline typically involves several steps (e.g., blocking, similarity estimation, clustering), with each step requiring its own configurations and tuning. The choice of the best configuration, among a vast number of possible combinations, is dataset-specific, as it has been shown experimentally, while it often requires the existence of some pre-labeled examples, i.e., a ground truth. In essence, finding the best configuration for resolving the entities of a dataset is a labor-intensive task not only for simple users that want their data cleaned, but also for ER experts. In this work, we introduce AutoER, a framework that automatically suggests the most promising ER configuration, even when a ground truth is not available. AutoER relies on sampling strategies that can significantly reduce the search space of configuration values, when some ground truth is available. When no pre-labeled examples of a given dataset are available, AutoER relies on a pre-defined set of ER-specific dataset features, along with configuration features, and other datasets that have an available ground truth, to train a regression model. We show experimentally that AutoER consistently and efficiently suggests near-optimal ER configurations, by comparing it to an exhaustive grid search, over eleven ER benchmark datasets.

SUBJECT AREA: Data Exploration

KEYWORDS: Entity Resolution, Auto Configuration, Artificial Intelligence

ΠΕΡΙΛΗΨΗ

Η ίδια οντότητα στον πραγματικό κόσμο (π.χ. μία ταινία, ένα εστιατόριο, ένα άτομο) μπορεί να περιγραφεί με διάφορους τρόπους σε διαφορετικά σύνολα δεδομένων. Η ανίχνευση οντοτήτων (Entity Resolution, ER) είναι το πρόβλημα της εύρεσης περιγραφών που αναφέρονται στην ίδια οντότητα, βελτιώνοντας έτσι την ποιότητα των δεδομένων και, κατ' επέκταση, την αξία τους. Ωστόσο, μια μεθοδολογία ER συνήθως περιλαμβάνει αρκετά στάδια (π.χ. αποκλεισμός, εκτίμηση ομοιότητας, ομαδοποίηση), με κάθε στάδιο να απαιτεί τη δική του διαμόρφωση και ρύθμιση. Η επιλογή των καλύτερων παραμέτρων και μεθόδων, ανάμεσα σε έναν τεράστιο αριθμό πιθανών συνδυασμών, είναι συγκεκριμένη για κάθε σύνολο δεδομένων, όπως έχει αποδειχθεί πειραματικά, ενώ συχνά απαιτεί την ύπαρξη κάποιων προεπισημασμένων παραδειγμάτων, δηλαδή των αληθινών ζευγαριών απο όμοιες οντότητες (ground truth). Ουσιαστικά, η εύρεση της βέλτιστης παραμετροποίησης, για την εύρεση των όμοιων οντοτήτων ενός συνόλου δεδομένων αποτελεί μια δύσκολη εργασία, όχι μόνο για απλούς χρήστες που θέλουν να καθαρίσουν τα δεδομένα τους, αλλά και για ειδικούς στο ER. Σε αυτή την διπλωματική εργασία, παρουσιάζουμε το AutoER, μια προσέγγιση που προτείνει αυτόματα την πιο υποσχόμενη μεθολογία ER, ακόμη και όταν δεν υπάρχει διαθέσιμο σύνολο αληθινά όμοιων ζευγαριών. Το AutoER βασίζεται σε στρατηγικές δειγματοληψίας που μπορούν να μειώσουν σημαντικά τον χώρο αναζήτησης των τιμών παραμετροποίησης, στο σενάριο όπου υπάρχει διαθέσιμο σύνολο αλήθειας. Όταν δεν υπάρχουν προεπισημασμένα παραδείγματα για ένα δεδομένο σύνολο δεδομένων, το AutoER στηρίζεται σε ένα προκαθορισμένο σύνολο χαρακτηριστικών του συνόλου δεδομένων που σχετίζονται με το ER, μαζί με χαρακτηριστικά της μεθοδολογίας και άλλα σύνολα δεδομένων που έχουν διαθέσιμο σύνολο αλήθειας, για να εκπαιδεύσει ένα μοντέλο παλινδρόμησης. Δείχνουμε πειραματικά ότι το AutoER προτείνει με συνέπεια και αποτελεσματικότητα σχεδόν βέλτιστες μεθοδολογίες ER, συγκρίνοντάς το με εξαντλητική αναζήτηση σε πλέγμα (Grid Search), σε έντεκα σύνολα δεδομένων αναφοράς για το ER.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Εξερεύνηση Δεδομένων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ανίχνευση Οντοτήτων, Αυτόματη Παραμετροποίηση, Τεχνητή Νοημοσύνη

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, George Papadakis, for his guidance and inspiration throughout our collaboration. He has been a true mentor, providing me with the opportunities to grow and take my first steps in the research field.

I would also like to extend my heartfelt appreciation to Professor Manolis Koubarakis for his inspiration, mentorship, and for giving me the opportunity to collaborate with him.

Additionally, I would like to acknowledge the co-supervision provided by Professor Vassilis Eythymiou from Harokopio University, as well as Professor Konstantinos Stefanidis from University of Tampere, Finland. This project was shaped and guided by all of their valuable contributions.

Funding. This research was funded by the Horizon Europe project STELAR (GA No. 101070122).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Entity resolution (ER) is the problem of identifying references to the same real-world object (e.g., a person, location, movie) in a dataset [14]. ER, as a data cleaning method, increases the quality and, subsequently, the value of the data, which can later be used further for downstream applications, like training a machine learning model. Apart from the benefits that it brings, ER also comes with a significant cost. Typically, ER pipelines involve several steps (e.g., blocking [51], meta-blocking, similarity estimation, clustering [27, 49]), with each step, in turn, involving several possible configurations, such as similarity threshold, several blocking strategies, language models, and clustering algorithms to choose from. As it has been repeatedly shown, there is no single configuration that dominates the others, as each ER setting (e.g., dataset characteristics, assumptions) has different requirements [14, 63, 34].

Even for end users who are ER experts, choosing the right configuration is a non-trivial task that involves a lot of trial-and-error experimentation. To make things worse, in a realistic setting, the end user does not have a ground truth of correct matches (otherwise, the ER problem is solved/not needed for that dataset) so they cannot even evaluate the performance of different configurations, other than a brief and unreliable manual inspection. Additionally, for a given dataset, predicting a successful configuration can be highly challenging. As illustrated in Figure 1.1, which presents the distribution boxplot of the ten primary datasets used in our evaluation, some datasets exhibit a narrow distribution. However, for the majority, such as D2, D3, D5, and particularly the most challenging dataset, D10, identifying an appropriate configuration proves to be significantly more difficult.

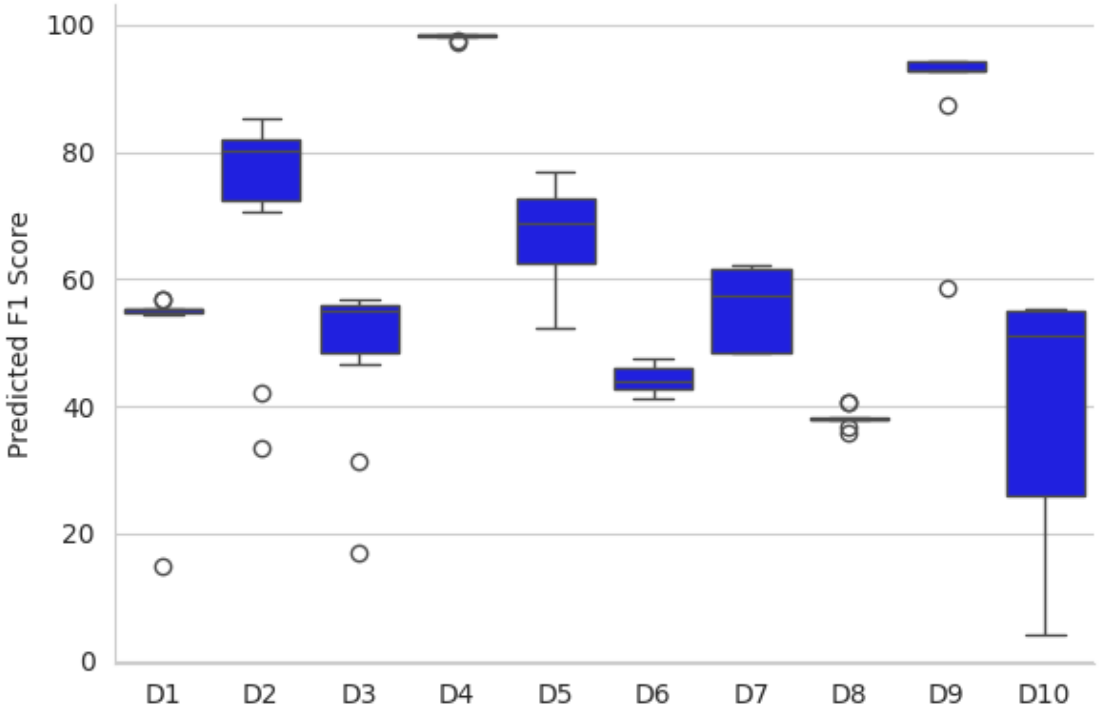

Figure 1.1: F1 distribution per dataset

In this thesis, we propose an auto-configurable ER pipeline that receives as input a dataset, optionally along with a very small number of labeled matches, and returns ER results after automatically deciding which setting is the most promising for that dataset. We experimentally show that in almost all cases our system yields results that are very close to the best obtainable results, identified via a grid search over all possible solutions.

In summary, the contributions of this work are the following:

- A self-configurable ER framework, AutoER, that automatically predicts the best ER pipeline configurations for a given dataset, with or without a known ground truth.
- AutoER can operate even without a ground truth, but it can also exploit a small set of labeled matches and quickly find the best configuration.
- Extensive experimental evaluation demonstrating the effectiveness and efficiency of our approach over 39,900 possible configurations tested against 10 widely used datasets.
- AutoER is open-source with a permissive license, available on Github: `https://github.com/AI-team-UoA/pyJedAI-AutoConfiguration`

Outline. The rest of this thesis is organized as follows. In Section 2, we describe the two variations of the problem that we tackle in this work. In Section 3, we discuss related works, highlighting the novelty of our approach. In Section 4.1, we present the overall end-to-end ER pipeline used in this work, for which we provide more details, depending on each of the two variations of the problem, in Sections 4.2 and 4.3. In Section 5, we present and discuss the experimental results. We conclude and present the future work of this thesis in Section 6.

2. PROBLEM DEFINITION

In this section, we describe the problem that this thesis tackles, which is end-to-end entity resolution, namely ETEER. We define two variations of this problem, depending on whether ground truth knowledge is available or not.

Let an *entity collection* $\mathcal{E}$ be a set of entity descriptions, where an entity description[1] is a set of attribute-value pairs representing a real-world entity (e.g., an entity description can be a database record, an ontology class instance, a row in a csv file, a json element).

End-to-end ER (ETEER) is the problem of receiving some entity collections and returning a set of entity *clusters*, with each cluster corresponding to set of entity descriptions that refer to the same real-world entity. Although ER can be generalized to receive as input one (Dirty ER [27]), two (Clean-Clean ER [49]), or more than two (Multi-Source ER [58, 57]) entity collections, in this work, we will assume the case of two clean (i.e., duplicate-free) entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$ that need to be resolved. Our definitions and methodology can be easily generalized to cover all other cases. More formally:

Definition 2.0.1 (End-to-end Entity Resolution (ETEER))**.** Given two entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, return a set of disjoint clusters $\mathcal{C}$, such that $\bigcup_{c_l \in \mathcal{C}} c_l = \mathcal{E}_1 \cup \mathcal{E}_2$, and $\forall c_l \in \mathcal{C}, \forall e_i, e_j \in c_l, e_i \equiv e_j$, where $e_i \equiv e_j$ denotes that the two entity descriptions $e_i \in \mathcal{E}_1$ and $e_j \in \mathcal{E}_2$ ***match***, i.e., refer to the same real-world entity.

This definition applies to Clean-Clean ER, also known as Record Linkage, where $\mathcal{E}_1$ and $\mathcal{E}_2$ are individually duplicate-free, but overlapping, sharing some entities. In that setting, each cluster contains either a single entity from one entity collection, or two entities, one from each collection. Definition 2.0.1 can be easily generalized to Dirty ER, where the input comprises a single entity collection with duplicates in itself. In this case, the resulting clusters are also disjoint, but there is no limit on their size, i.e., the number of entities they contain. It can also be generalized to the case of Multi-Source ER, requiring that each cluster cannot contain more than one entity from the same collection.

The problem of Definition 2.0.1 is typically solved through an *ETEER pipeline*, denoted by $w(\mathcal{E}_1, \mathcal{E}_2)$, i.e., a series of processing steps for solving ER, with each processing step potentially requiring the configuration of some parameters.

Definition 2.0.2 (ETEER Pipeline)**.** An ETEER Pipeline $w(\mathcal{E}_1, \mathcal{E}_2) = (S, P, V)$ is a series of processing steps $S = s_1, s_2, \ldots, s_n$ for solving ETEER, with each processing step s_i potentially requiring the configuration of some parameters $P_i = p_1^i, p_2^i, \ldots, p_m^i$, and with each parameter p_j^i accepting a set of possible values V_j^i.

Following the previous definition, we will sometimes simply call the set $P = \bigcup_{i \in [1..n]} P_i$ as *configuration parameters* and we will also simply call the set $V = \bigcup_{P_i \in P, p_j \in P_i} V_j^i$ as *configuration values* or *possible values*.

The effectiveness of an ETEER pipeline is assessed in terms of precision, recall, and F1-score, with respect to a ground truth of known matches $\mathcal{D}$. In more detail, precision is the number of clusters in $\mathcal{C}$ which contain a pair of entities that also appears in $\mathcal{D}$; recall is the portion of pairs in $\mathcal{D}$ that co-occur in some cluster of $\mathcal{C}$; F1-score is the harmonic mean of precision and recall. The time efficiency of an ETEER pipeline can be measured in terms

[1] For brevity, we will also refer to entity descriptions, simply as "entities".

of the overall runtime, i.e., the time between receiving $\mathcal{E}_1$ and $\mathcal{E}_2$ and producing $\mathcal{C}$. Given that we are interested in finding the best configuration for a fixed ETEER pipeline, we will focus more on measuring time efficiency of AutoER in terms of *search time*, i.e., the time needed to produce a the suggested configuration values.

In this context, we address two problems, based on whether a ground truth is available (Problem 1) or not (Problem 2) for configuration tuning. Such a (subset of the) ground truth is also known as "seed alignment" that is used for training entity alignment models [21, 63]. For the evaluation, we always use a ground truth of known matches. In Problem 1, we also use it for tuning the configuration parameters, while in Problem 2, we do not use any portion of the ground truth.

Problem 1. Given an ETEER pipeline $w(\mathcal{E}_1, \mathcal{E}_2)$ for two entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, along with a subset of the ground-truth of matches $\mathcal{D}' \subseteq \mathcal{D}$, return the configuration values V of $w(\mathcal{E}_1, \mathcal{E}_2)$ that maximize the effectiveness of $w(\mathcal{E}_1, \mathcal{E}_2)$ in terms of F1-score, while minimizing the search time.

The naïve approach is to apply *grid search*, considering all meaningful values for all configuration parameters in the specified pipeline. However, this is impractical and time-consuming, even for simple ETEER pipelines, because they typically involve numerous configuration parameters with large domains. Therefore, more advanced algorithms for configuration optimization are required to effectively reduce the extremely large search space of configuration parameters.

A more difficult variation of the first problem is to fine-tune a specific ETEER pipeline without having any portion of the real matches $\mathcal{D}$ provided as input. This task can be formalized as follows:

Problem 2. Given an ETEER pipeline $w(\mathcal{E}_1, \mathcal{E}_2)$ for two entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, return the configuration values V of $w(\mathcal{E}_1, \mathcal{E}_2)$ that maximize the effectiveness of $w(\mathcal{E}_1, \mathcal{E}_2)$ in terms of F1-score, while minimizing the search time.

To the best of our knowledge, no prior work focuses on this task.

3. RELATED WORK

In [19], we envisioned a self-configurable, end-to-end ER framework that can operate in different domains, focusing on data interlinking, while also supporting ethical constraints, like fairness. As a follow-up to this work, we are now providing a practical implementation of many of those envisioned features, leaving out domain adaptations and ethical considerations, as they are orthogonal to the topic of this work. This section will make a thorough pass of the area we are researching for, Entity Resolution. Next, we will present the current advancements in AutoML and finally we will delve into the intersection of these areas that form the target area of this study.

Entity Resolution (ER) is a key research area within Data Management and Data Exploration, gaining significant attention over the last four decades. ER has a crucial role in addressing the challenges posed by the vast and complex nature of data, including knowledge graphs, ontologies, and structured or semi-structured data. To establish meaningful links between entities in such datasets, a variety of techniques have been developed, focusing on four primary dimensions: Veracity, Volume, Variety, and Velocity. As outlined in our recent tutorial [44], ER has evolved through five distinct generations, each addressing different aspects of these challenges.

The **1st ER generation** focused on tackling Veracity, through end-to-end pipelines consisted of three consecutive steps: *Schema Matching*, *Blocking*, and *Entity Matching*. Thats the first pipeline-like approach presented for ER. We should mention that like the traditional Machine Learning (ML) pipelines, that consist of separate building blocks, (i.e Data Processing, Feature Selection, Model Selection, Hyperparameter tuning) an analogous approach has been suggested also for ER. The first step is Schema Matching that tries to match the relevant attributes across two structured datasets with different schemata [9, 40]. Researchers at this point discovered the most challenging problem of ER. Performing comparisons between every entity and all others leads to a quadratic time complexity of $O(n^2)$ [13]. As a response to the complexity inherent in ER tasks, various algorithms and techniques have been developed to reduce the computational overhead, primarily by minimizing the number of comparisons and similarity evaluations. One such technique is known as *Blocking*, which aims to reduce computational cost by restricting comparisons to entity profiles deemed most similar. This is achieved by generating signatures, composed of selected portions of attribute values, particularly those representing the most informative attributes. Subsequently, a method for similarity assessment between entity pairs within the same block is incorporated into the ER pipeline. This step, referred to as *Entity Matching*, determines whether two entities represent a match, a non-match, or are classified as uncertain [13]. However, with the increasing demand for ER, the need to process larger and more complex datasets has escalated, thus presenting significant challenges in scaling ER to accommodate Big Data environments. In this thesis, we will work with a pipeline, that shares in commom some of the building blocks that were introduced in the very first generation.

The **2nd ER generation** addressed challenges related to Volume and Veracity by extending the principles of the 1st generation to Big Data. Techniques such as parallelization and Map/Reduce [15] were employed to handle blocks in parallel after the Blocking phase, significantly reducing time complexity [32]. The same approach was applied to Entity Matching [10], although this introduced load balancing challenges [33]. With the **3rd ER Generation**, the focus shifted to tackling Variety, Volume, and Veracity, where heterogeneity and noisy data posed additional challenges. Schema Clustering [24] was introduced

as the first step in the ER pipeline to group schemas based on attribute values rather than semantics, enhancing precision without sacrificing recall. Block Building and Block Processing were then introduced to create and refine blocks without schema alignment or manual intervention. Entity Matching generated a similarity graph, where nodes represent entities and edges denote similarity scores [36, 37, 62], leading to the final optimization step, Entity Clustering [27]. This is another building block of the auto configuration we suggest for this task. The **4th ER Generation** emerged to address new challenges involving Velocity, alongside Variety, Volume, and Veracity. As datasets became more dynamic and streaming in nature, due to the evolving web and increasing applications, the speed and cost of performing ER required optimization. To reduce the number of comparisons and resource usage, Progressive ER introduced a pay-as-you-go model, prioritizing the most valuable comparisons within a given budget. Another advancement, Incremental ER [25], aimed to minimize update costs in streaming use-cases, such as those provided by web-based REST APIs. Additionally, Query-driven ER [2] was developed to progressively resolve entities in response to incoming queries over time [28].

Finally, the last generation, the **5th ER Generation**, contains the latest advancements in ER. The most significant additions and innovations are relevant to the Deep Learning growth, and the changes in the ML field that it brought. Nowadays, more and more applications use pre-trained Language Models (LMs) [16, 55], utilize transformers [65], and the most recent advancements like Large Language Models (LLMs)[12, 11]. Our work will focus on a pipeline, created for ER that utilizes both pre-trained word- and sentence- LMs.

ER Tools. Several ER tools are designed to require minimal or no configuration. Some of the most hands-off approaches are described in papers like [38, 43, 68]. For example, Ditto [38] extends the BERT-based EMTransformer by integrating external, domain-specific knowledge, such as Part-of-Speech (POS) tagging, and employing data augmentation techniques to create synthetic training instances. However, Ditto requires configuration of several deep learning parameters, such as learning rate and epochs. Similarly, DeepMatcher [43] is a comprehensive framework for deep learning-based matching algorithms, supporting various pre-trained static embeddings, including GloVe and FastText, with FastText as the default choice. While DeepMatcher involves fewer configuration parameters, users still need to specify word embeddings, and no auto-configuration component has been proposed. Finally, ZeroER [68] is an unsupervised-learning approach, used for ER with almost none parameters to be configured.

In this work, we will use one of the most recent tools named pyJedAI [45], that integrates the most recent advancements described in the 5th generation. pyJedAI is an open-source framework designed for Clean-Clean ER (CCER), Dirty ER (DER), and Geospatial Interlinking, offering a wide range of learning-free methods since its development began in 2017 [50, 53, 45]. Released under the Apache License 2.0, pyJedAI is format-agnostic, enabling ER on structured, semi-structured, and unstructured datasets, while supporting diverse scenarios where different formats coexist. Its extensible architecture allows seamless integration of algorithms targeting specific workflow steps by using concise interfaces and wrapper mechanisms. For this reason, we selected this tool as the basis of this study. This approach has been successfully applied to integrate advanced techniques such as FAISS [30], FALCONN [3], DeepBlocker [64], and language models like (Sentence)BERT [69]. pyJedAI is holistic in nature, supporting schema-based and schema-agnostic pipelines, and offers both budget-agnostic (batch) and budget-aware (progressive) execution modes. Furthermore, it is highly efficient in terms of time and memory, utilizing optimized Python data structures while supporting all major execution modes, including single-core, multi-core, and massively parallel execution via Apache Spark.

AutoML. Automated Machine Learning (AutoML) has gained significant attention in recent years due to advancements in machine learning (ML) and the increasing demand for hands-free systems to support both developers and novices in efficiently creating ML applications. AutoML systems automate the process of selecting and optimizing ML pipelines, which are essential as different datasets often require distinct configurations. Leading AutoML systems, such as Auto-WEKA [35], Auto-sklearn [22], TPOT [47], and Auto-Keras [29], optimize across preprocessors, classifiers, and hyperparameters, thereby reducing the manual effort involved in pipeline creation. AutoML has evolved through challenges such as the AutoML competitions by ChaLearn [26], which systematically evaluate AutoML systems under strict time and resource constraints. These systems support dynamic pipeline construction, including processes like data preparation, feature engineering, model generation, and evaluation. Model generation involves both traditional ML models (e.g., SVM, KNN) and neural architectures, with optimization techniques classified into hyperparameter optimization (HPO) and architecture optimization (AO). Neural architecture search (NAS) is a key component, focusing on automating the design of neural network architectures [20], further enhancing the capabilities of AutoML in democratizing machine learning by making it accessible and efficient for users with limited ML expertise.

HPO Frameworks. A number of toolkits have been developed for auto-configuring tasks, such as Optuna [1], SMAC [39], Hyperopt [6], Autotune [31], and Vizier [61, 23]. These frameworks utilize various algorithms for parameter sampling, which is the process of selecting combinations of parameters from a predefined search space. For instance, Spearmint [59] uses Gaussian Processes, while Hyperopt employs the tree-structured Parzen estimator (TPE). SMAC [39] introduced a method based on random forests. More recent frameworks, such as Google Vizier [61, 23] and Tune [31], incorporate pruning algorithms. These algorithms assess the intermediate outcomes of each trial and terminate those that seem less promising, thereby accelerating the search process. This is akin to the early-stopping technique commonly used in various machine learning tasks. For this study, and to construct the ETEER methodology, Optuna toolkit was selected for the HPO task.

Auto-ER Approaches. There has been limited research on auto-configuring ER pipelines and ER problems in general. Most existing work on auto-configuring ER focuses on the Entity Matching step. For instance, the study in [70] introduces a transfer-learning framework that utilizes pre-trained EM models derived from extensive knowledge bases (KBs). However, this approach is domain-specific, heavily relying on relevant and well-curated KBs, which limits its applicability in real-world scenarios where such KBs are unavailable. In contrast, the ETEER methodology is domain-agnostic and can be applied across various data types. Another study [48] attempts to construct a pure AutoML Entity Matching pipeline using auto-sklearn. Their approach differs from ours as auto-sklearn is employed to predict whether a pair of entities is a match/non-match by transforming text entities through BERT-based pre-trained models. Lastly, [66] proposes the AutoML-EM framework, which shares similarities with our approach by using AutoML models to predict the optimal Random Forest configuration. However, this work is limited to Random Forest models and relies on an active learning and self-training strategy, which necessitates a human-in-the-loop algorithm when ground truth pairs are scarce.

4. METHODOLOGY

4.1 ETEER pipeline

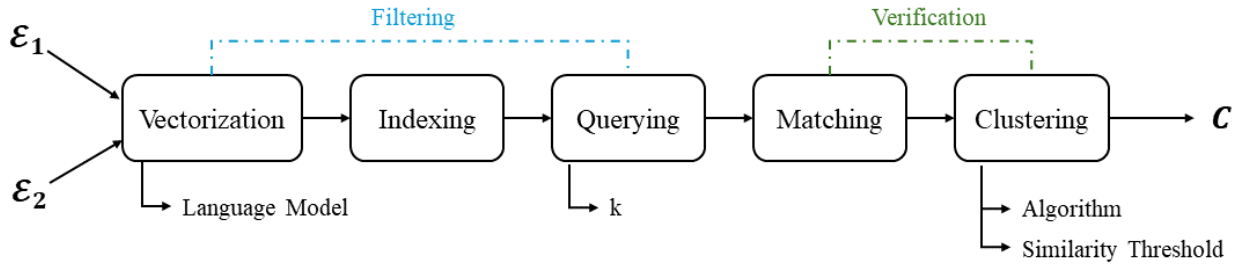

Figure 4.1: The ETEER pipeline considered in this work.

In this section, we present the ETEER pipeline that will be used in the context of Problem 1 and Problem 2. In general, ETEER pipelines consist of two steps, due to the quadratic complexity of the brute-force approach:

- *Filtering*, which reduces the computational cost by identifying a set of promising *candidate pairs* to be matched.
- *Verification*, which analytically examines each candidate pair.

Based on recent experimental studies [69], our solution relies on the ETEER pipeline that is shown in Figure 4.1, which leverages language models. This approach not only combines high effectiveness with high time efficiency, but also requires the fine-tuning of a limited set of configuration parameters.

The first step is Vectorization, where the input entities are transformed into embedding vectors using pre-trained LMs, either static ones like Word2Vec [41] and GloVe [54], or dynamic ones, like BERT [17] and SentenceBERT [56]. The former are context-agnostic, assigning each word to the same vector regardless of its meaning, while the latter are context-aware, generating different embeddings for different meanings of the same word (e.g., trunk as part of a tree or an elephant). Therefore, selecting the appropriate language model is crucial for achieving good results. Following [69], we restrict our configurations to five state-of-the-art SentenceBERT models and two static models, listed in Table 4.1 **(Parameter: LM)**. Describing them in more detail goes beyond the scope of this work.

Note that we follow the schema-agnostic procedure, which simply concatenates all attribute values into a sentence describing each entity. This way, we inherently address noise in the form of misplaced values, while also achieving very high performance under versatile settings [69].

The second step is Indexing, which receives as input all embedding vectors of $\mathcal{E}_1$ and organizes them in a data structure that facilitates their retrieval in descending distance from a given query (i.e., an embedding vector of $\mathcal{E}_2$). Based on a recent experimental study [4], we employ FAISS [18] for this purpose, due to its excellent balance between effectiveness and time efficiency. No parameter needs to be configured in this step, given that we employ the configuration returning the exact results.

The third step is Querying, which receives as input all embedding vectors of $\mathcal{E}_2$ and uses each vector/entity as a query to the FAISS index. The result of each query consists of the

Table 4.1: Configuration parameters and their possible values.

Parameters (P)	**Possible values (V)**
Language model (LM)	smpnet, st5, sdistilroberta, sminilm, sent_glove, fasttext, word2vec
k	$[1, 100]$ with a step of 1
Clustering algorithm	Unique Mapping Clustering, Kiraly Clustering, Connected Components
Similarity Threshold	$[0.05, 0.95]$ with a step of 0.05

k most similar entities from $\mathcal{E}_1$ in terms of cosine similarity. The k parameter is the sole configuration parameter of this step, playing a major role in the performance of ETEER. The higher the value of k, the higher the filtering recall, at the cost of lower precision. Note that the recall of this step sets the upper limit of the overall ETEER recall, given that the subsequent steps do not detect new matches. Hence, k serves as the second parameter to the selected ETEER **(Parameter: k)**.

Note that together, the first three steps implement the Filtering approach, which reduces the computational cost to the k most similar entities from $\mathcal{E}_1$ per entity from $\mathcal{E}_2$. This means that it receives as input the given entity collections and returns as output as set of candidate pairs P.

The set of candidate pairs P is then transformed into a *similarity graph*, where every node is an entity and every edge represents a candidate pair, weighted according to the similarity score returned by FAISS. The similarity graph is bipartite in the Record Linkage settings we are considering. This transformation is carried out by Matching, the fourth step of our ETEER pipeline. No configuration parameter is involved in this process.

The final step of the ETEER pipeline is Clustering, which applies bipartite graph matching algorithms to the similarity graph generated by the previous step. Based on a recent experimental study [49], the three state-of-the-art algorithms listed in Table 4.1 are supported **(Parameter: Clustering)**. All algorithms are configured by a similarity threshold **(Parameter: Threshold)**, which prunes edges with a lower weight.

The configuration space of our pipeline is summarized in Table 4.1. Note that grid search should consider 39,900 different combinations on each dataset in order to maximize F1 (7 LM options $\times$ 100 k options $\times$ 3 clustering options $\times$ 19 threshold options). Our goal is to develop and apply algorithms that converge to the maximum performance, while minimizing the trials in this search space.

4.2 Tackling Problem 1

Fine-tuning an ETEER pipeline based on the ground truth comprising all duplicate pairs is similar to hyperparameter fine-tuning in ML and DL models. The only difference is the form of the data: in the latter case, the data comes in the form of positive and negative labelled instances, which is not possible in the former case, due to the extremely large number of pairs that stem even from relatively small datasets with few thousand entities in $\mathcal{E}_1$ and $\mathcal{E}_2$. Besides, the number of positive pairs is linear with respect to the number of given entities, while all other pairs are negative. Therefore, instead of enumerating and labelling all possible pairs in the Cartesian product of $\mathcal{E}_1 \times \mathcal{E}_2$, hyperparameter fine-tuning is guided by the F1 score corresponding to the matches generated by each configured

ETEER pipeline. This is the only change that can be applied to existing algorithms for hyperparameter fine-tuning, yet they have not been applied before to Problem 1.

More specifically, hyperparameter fine-tuning is typically carried out by *sampling*, which tries to maximize two goals of auto-configuration [1]: (i) the efficiency in the configuration strategy, which is the process of determining the set of parameters that should be investigated, and (ii) the efficiency of performance estimation strategy, that estimates the value that a set of configurations will produce, based on learning curves, and finally determines the set of parameters that should be discarded.

Sampling can be categorized in two main types: (i) *relational sampling* is the sampling that exploits the correlations among the parameters, and (ii) *independent sampling* that samples each parameter, independently. The former are not compatible with categorical variables. As a result, we consider the following four state-of-the-art sampling algorithms, three are independent and one is relational (assume that d is the dimension of the search space and n is the number of finished trials):

1. *RandomSampler* [8], with complexity of $O(d)$, is an independent sampling technique, where hyperparameter values are chosen randomly from the specified search space. This method does not use any prior knowledge or adaptive mechanisms to guide the search process. Instead, it relies on randomness to explore the search space. Random sampling achieves high efficiency in many cases because each point in the search space has an equal probability of being explored. This approach results in a more uniform coverage of the search space. Even in high-dimensional spaces, random sampling is more likely to identify near-optimal configurations with fewer trials, especially when compared to the grid search approach.

2. *TPESampler*, with complexity of $O(d * n * log_2 n)$, uses the TPE (Tree-structured Parzen Estimator) algorithm and utilizes independent sampling [67, 7, 5]. During each trial, for every parameter, the TPE algorithm fits two Gaussian Mixture Models (GMMs): $l(x)$ to the set of parameter values with the best objective outcomes, and $g(x)$ to the remaining parameter values. It then selects the parameter value x that maximizes the ratio $l(x)/g(x)$. In other words, TPESampler models the distribution of good and bad hyperparameters using non-parametric density estimators, focusing on promising regions of the search space. By iteratively updating these models based on observed results, TPESampler needs more trials to explore potentially optimal areas, but has improved search efficiency compared to random or grid search.

3. *QMCSampler* (Quasi Monte Carlo Sampler) [8], with complexity of $O(d * n)$, uses quasi-random methods, specifically low-discrepancy sequences, to enhance the efficiency of hyperparameter optimization by ensuring uniform coverage of the search space. Unlike random search, which can have uneven distribution, low-discrepancy sequences spread points more evenly, avoiding clumps and gaps. This uniformity increases the probability of finding optimal hyperparameters with fewer trials, particularly in high-dimensional settings where important dimensions need adequate sampling.

4. *GPSampler* (Gaussian Process-based bayesian Sampler) [60, 8], with complexity of $O(n^3)$, is a relational sampler that fits a Gaussian process to the objective function and optimizes the acquisition function to suggest the next set of parameters. Bayesian optimization methods, in general, first construct a probabilistic model of

the objective function and then exploit this model to estimate the most promising regions in the search space. The GPSampler uses a Gaussian process to represent the probabilistic distribution of the objective function, providing flexibility and adaptability. Among various acquisition functions, it employs Log Expected Improvement (logEI) in combination with Quasi-Monte Carlo (QMC) sampling for optimizing the acquisition function. Gaussian process-based Bayesian approaches, have proven highly effective in hyperparameter optimization (HPO) problems, as they leverage the dependencies between hyperparameters and avoids redundant trials, due to their probabilistic estimation of the objective function.

To the best of our knowledge, none of these algorithms has been applied to fine-tuning ETEER pipelines.

4.3 Tackling Problem 2

As explained above, the sampling algorithms are guided by the performance that corresponds to each tested configuration. In the context of Problem 2, though, this approach is inapplicable, due to the lack of a complete ground truth for the dataset at hand. Nevertheless, ETEER fine-tuning can still be modelled as a learning-based task by leveraging the ground truths from other datasets.

More specifically, we frame our solution to Problem 2 as a regression task: first, we define a set of features that capture the characteristics of a Record Linkage dataset. Then, these *dataset features* are concatenated with four *configuration features*, which stand for the four configuration parameters of our ETEER pipeline in Figure 4.1. This means that each feature vector combines parameter configurations with general characteristics of a particular dataset. For each feature vector, we apply the respective parameter configuration to a particular dataset with known ground truth in order to estimate the corresponding target variable, i.e., the corresponding F1 score. Applying numerous ETEER configurations to multiple datasets with known ground truth yields a sizeable training set that can be used for learning a regression model. The learned model is then applied to the feature vectors extracted from the given dataset, which lacks a ground truth. Again, there is a different feature vector for each parameter configuration we want to try. The one yielding the highest predicted F1 score is selected as the optimal configuration for the dataset at hand. To put this idea into approach, we need to:

1. define the features capturing general characteristics of an ER dataset,
2. define the approach for generating the parameter configurations to be tested,
3. choose the datasets with known ground truth that will generate the training instances, and
4. apply a learning procedure for training a prediction model.

For Point 3, we choose a set of 11 established datasets for Record Linkage that are publicly available and accompanied by a complete ground truth. See Section 5 for more details. We delve into Points 1, 2 and 4 in the following.

4.3.1 Dataset Features

The dataset features used by our approach should adhere to the following principles: (i) They should be *generic*, applying seamlessly to any ER dataset, regardless of the corresponding flavor of ER (i.e., Record Linkage, Deduplication or Multi-source ER). (ii) They should be *effective*, capturing all aspects of a dataset that might affect the performance of an ETEER applied to it. (iii) They should be *efficient*, involving low extraction cost and overhead, so that it is possible to extract predictions for numerous feature vectors from the learned model. In this context, we define the following dataset features:

1. Number of entities, i.e., the total number of entities in the dataset ($|\mathcal{E}_1| + |\mathcal{E}_2|$ in the case of Record Linkage datasets).
2. Number of attributes, i.e., the total number of distinct attributes describing all given entities.
3. Number of distinct values, i.e., the total number of distinct values across all attributes.
4. Number of attribute-value pairs, i.e., the total number of attribute-value pairs in the dataset.
5. Mean attribute-value pairs per entity, i.e., the average number of attribute-value pairs per entity.
6. Mean attribute-value pairs per attribute, i.e., the average number of attribute-value pairs per attribute.
7. Number of missing attribute-value pairs, i.e., the total number of missing attribute-value pairs in the dataset.
8. Mean distinct values per attribute, i.e., the average number of distinct values per attribute.
9. Mean total attribute value length per entity, i.e., the average total length of attribute values per entity.
10. Mean attribute value tokens per entity, i.e., the average number of tokens in attribute values per entity.
11. Mean distinct values per entity, i.e., the average number of distinct values per entity.
12. Max values per entity, i.e., the maximum number of values for any single entity.

These dataset features are concatenated with four configuration features, as illustrated in Figure 4.2.

4.3.2 Instance Generation

We follow three different procedures for generating a set of labelling instances from each dataset:

1. Grid search, which applies all configurations in Table 4.1 to extract the corresponding F1 score.

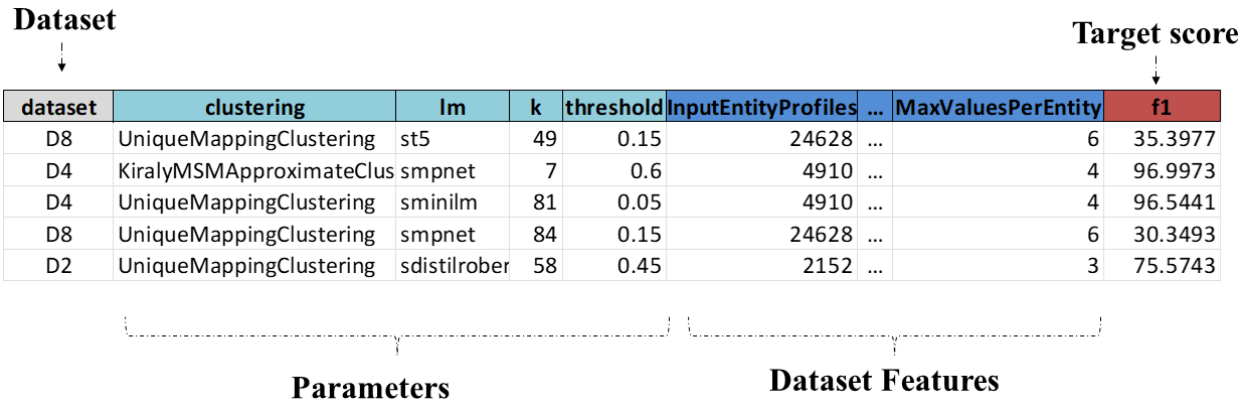

Figure 4.2: Joining dataset features with ETEER parameters.

2. Sampling-based search, which applies the three sampling methods described in Section 4.2.

3. All, which combines the instances generated by grid and sampling-based search.

Among these methods, grid search is expected to involve a much more time-consuming process, due to the larger number of configurations that it typically considers. However, the actual run-time of the configurations tested by grid search might be much lower than that of the configurations proposed by sampling-based search. Most importantly, instance generation is an offline process that does not affect the prediction time for the learned model. Therefore, we are mostly interested in the effectiveness of the models learned by each instance generation approach. This is experimentally assessed in Section 5.

4.3.3 Learning procedure

So far, we described the process for assembling a large set of labelled instances that associate the dataset and configuration features with the corresponding F1. This process is applied to datasets D1-D10 described in Section 5.1 except for the dataset is given as input to be fine-tuned (in a vein similar to leave-one-out cross-validation). For this dataset, we feed all feature vectors to the trained to retrieve the predicted F1 score per configuration. To train the prediction model, we consider two learning approaches:

1. *Auto-ML.* Instead of manually testing several state-of-the-art regression algorithms, we leverage Auto-ML to automatically train a high-performing regression algorithm.

2. *Individual regressors.* The following regression algorithms were considered: Linear Regression, Lasso, Ridge, Random Forest Regressor, Gradient Boosting (XGBoost) Regressor as well as a Neural Network (NN). The neural network, defined using PyTorch, is designed with adjustable parameters including the size of the hidden layer, learning rate, and number of epochs. Optuna [1] conducts a series of trials (50 in our setup) to identify the optimal hyperparameters that minimize the validation loss. The neural network architecture consists of an input layer, one hidden layer with ReLU activation and dropout for regularization, and an output layer. The Adam optimizer is used for training, and a learning rate scheduler adjusts the learning rate based on the validation loss. Early stopping is implemented to halt training when the validation loss no longer improves, ensuring the model does not overfit.

Hyperparameter optimization is performed using the TPESampler algorithm, which efficiently explored the configuration space we defined for each algorithm. In fact, the fine-tuning minimized the mean squared error on a validation set derived from 10% of a stratified random sample from the training data. After determining the best hyperparameters, the model is retrained on the full training set and used to predict the F1-scores of the test set. This process is applied independently to each dataset. The only exception is Linear Regression, which is a parameter-free approach.

The training data was preprocessed by scaling and converting categorical variables into dummy variables to maintain consistency across all datasets.

5. EXPERIMENTS

In this section, we first present our experiments for tackling Problem 1 (Section 4.2), and then by utilizing all the results produced we proceed on showing the results for tackling Problem 2 (Section 4.3).

5.1 Experimental setup

All experiments were implemented in Python, v. 3.9. For the implementation of the ETEER pipeline, we used pyJedAI v. 0.1.8. For the implementation of the regression algorithms, we used scikit-learn v. 1.4.2, and for the sampling-based configuration search algorithms we used Optuna v. 3.6.1. To implement the AutoML process, we used auto-sklearn with the following parameters:

- Time limit in seconds for the search of appropriate models (time_left_for_this_task) equal to 24 hours.
- Time for a single call in the ML model (per_run_time_limit) equal to 2 hours.
- Memory limit in MB for the machine learning algorithm (memory_limit) equal to 6144*4 MB.
- The number of jobs to run in parallel (n_jobs) was set to 1.

No validation set used on this task, as auto-sklearn creates one internally.

All AutoML experiments were executed on a server running Ubuntu 22.04, equipped with Intel Xeon E5-4603 v2 @ 2,2GHz and 16 GM RAM. All Optuna, Individual Regressors approach and pyJedAI experiments were executed on a server running Ubuntu 22.04, equipped with Intel Core i7-9700K @ 8x 4,9GHz and 68 GB RAM, D11 experiment was executed in the same server.

Evaluation. The evaluation metrics include mean squared error (MSE), the ratio of the best-predicted F1-score to the *global maximum F1*-score. After exploring all Optuna and Gridsearch trials, the maximum F1-score per dataset, serves as the global best F1-score (GB-F1). Also, for the evaluation of the predicted configurations we measured the difference between the predicted and actual best scores. Specifically, this metric will be found later as *F1-Ratio* and it is the fraction of ETEER predicted configuration F1 divided by the GB-F1 for each dataset.

$$\text{F1-Ratio}_d = \frac{\text{F1}_{\text{predicted},d}}{\text{GB-F1}_d}, \quad \forall d \in D$$
$$\text{GB-F1}_d = \max_{t \in T_d} (\text{F1}_d(t)) \tag{5.1}$$

Where T_d are all trials done $\forall d \in D$ and $D = \{D1, D2, ..., D10\}$.

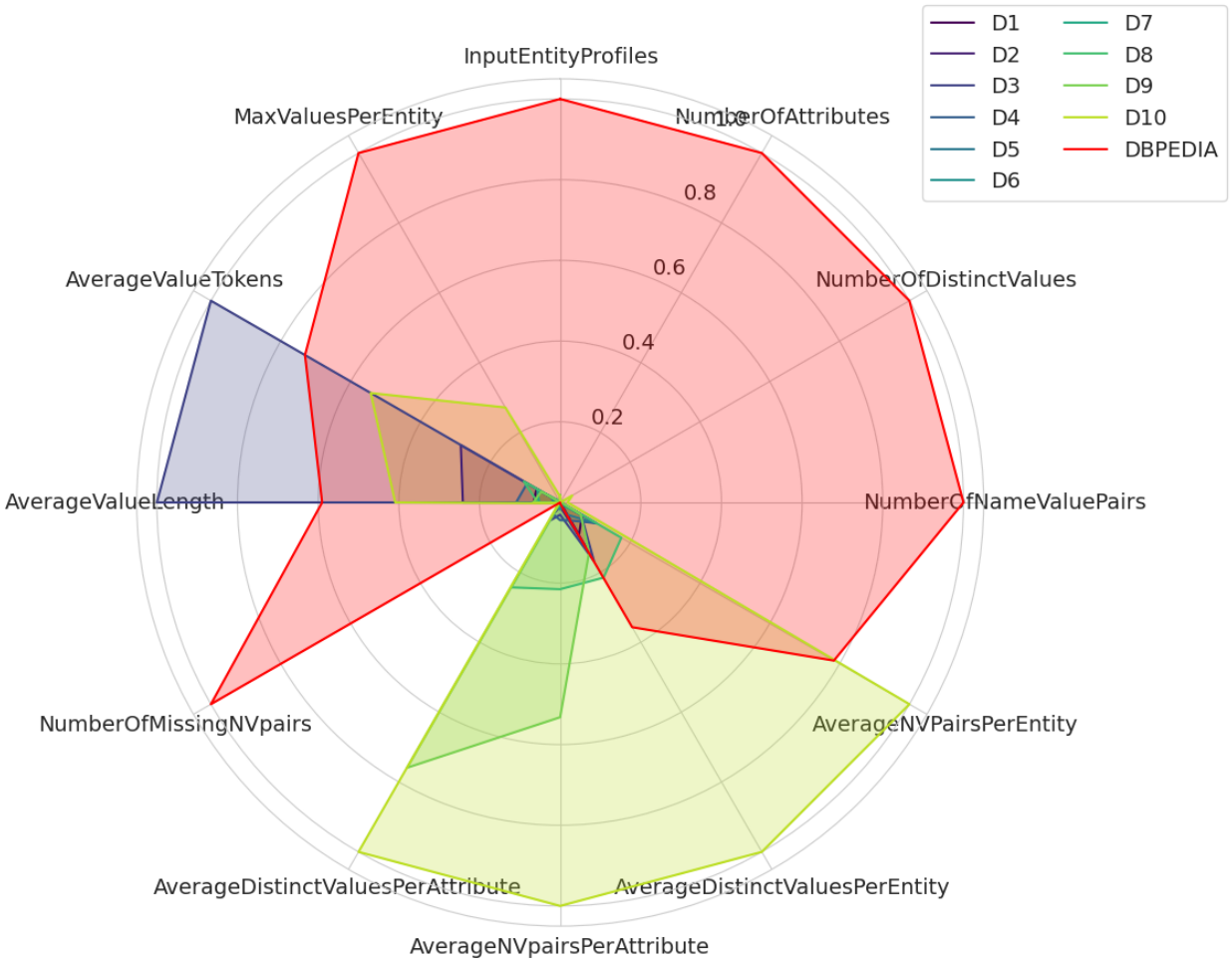

Figure 5.1: All dataset features comparison.

Datasets. For Clean-Clean ER, we use 11 real-world datasets that are popular in the literature [64, 42, 34, 52]. Their technical characteristics are reported in Table 5.1, and all datasets are publicly available[1]. D_1, which was first used in OAEI 2010[2], contains restaurant descriptions. D_2 encompasses duplicate products from the online retailers Abt.com and Buy.com [34]. D_3 matches product descriptions from Amazon.com and the Google Base data API (GB) [34]. D_4 entails bibliographic data from DBLP and ACM [34]. D_5, D_6 and D_7 involve descriptions of television shows from TheTVDB.com (TVDB) and of movies from IMDb and themoviedb.org (TMDb) [46]. D_8 matches product descriptions from Walmart and Amazon [42]. D_9 involves bibliographic data from publications in DBLP and Google Scholar (GS) [34]. D_{10} interlinks movie descriptions from IMDb and DBpedia [50], including a different snapshot of IMDb than D_5 and D_6. Finally, D11 matches two different versions of DBpedia that chronologically differ by 3 years and will be used as a scalability test in our evaluation. Figure 5.1 explores the dataset features extracted from each dataset, illustrating that D2 through D6 exhibit similar feature values, whereas D8, D9, D10, and particularly D11 differ significantly, with D11 representing a completely distinct feature area.

[1] `https://zenodo.org/records/10059096`
[2] `http://oaei.ontologymatching.org/2010`

Table 5.1: Datasets used in our experimental analysis.

Dataset	Names	S1	S2	#Duplicates
D1	Restaurants1-Restaurants2	340	2,257	89
D2	Abt-Buy	1077	1,076	1,076
D3	Amazon-Google Products	1,355	3,040	1,103
D4	DBLP-ACM	2,617	2,295	2,225
D5	IMDB-TMDB	5,119	6,057	1,969
D6	IMDB-TVDB	5,119	7,811	1,073
D7	TMDB-TVDB	6,057	7,811	1,096
D8	Walmart-Amazon	2,555	22,075	853
D9	DBLP-Google Scholar	2,517	61,354	2,309
D10	IMDB-DBpedia	27,616	23,183	22,864
D11	DBpedia	1,190,734	2,164,041	892,579

5.2 Experiments with Available Ground-truth [Problem 1]

We evaluated the performance of Optuna using the four available samplers discussed in Section 4.2: *TPESampler*, *RandomSearchSampler*, *QMCSampler*, and *GPSampler*. Optuna was configured to run 100 trials per dataset, adhering to the recommendations from the Optuna documentation[3]. To analyze the convergence behavior of each sampler, we performed experiments with five different random seeds, varying the maximum number of trials from 5 to 100 in increments of 5. The objective was to determine the number of trials needed to find a near-optimal solution. Figure 5.2 illustrates the convergence analysis of each sampler across all datasets. The y-axis shows the average of the maximum F1-scores achieved per sampler. This ”maximum” refers to the highest F1-score within each set of trials (5, 10, ..., 100), and the ”average” is computed across five different random seeds, providing an average of these maximum scores. The x-axis represents the different sets of trials evaluated.

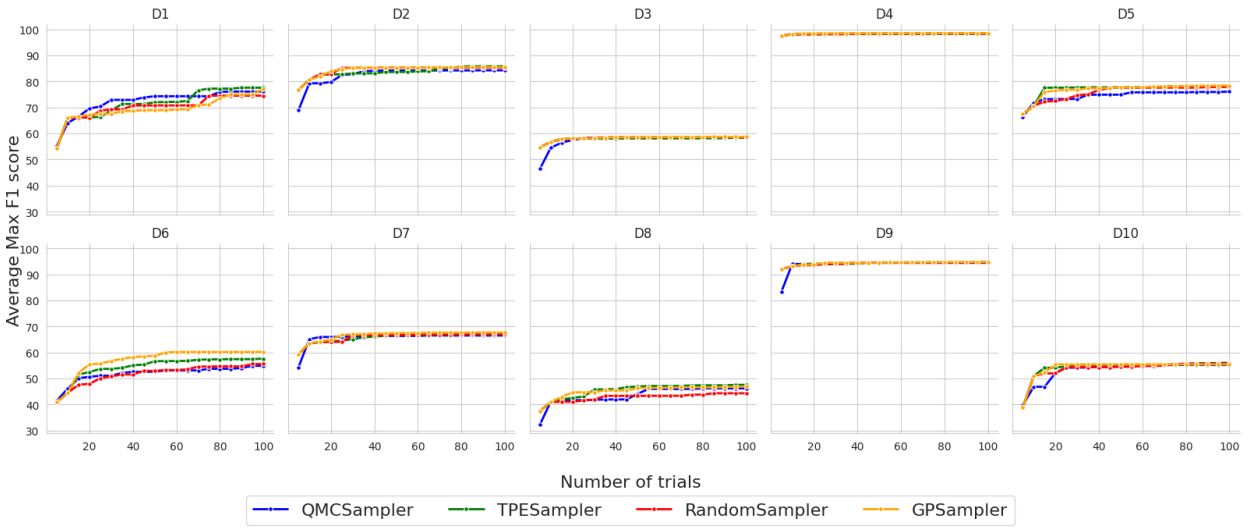

Figure 5.2: Samplers F1 convergence.

The number of trials in Optuna is a user-defined parameter, and Optuna will execute exactly the number of trials specified. Convergence was generally observed around 20 trials

[3] https://optuna.readthedocs.io/en/stable/reference/samplers/index.html

for most datasets, except for D1, D6, and D8. Consequently, with 20 trials, the ETEER methodology can offer a sufficiently optimal solution when a ground truth is available. Additionally, the figure indicates that there is no significant performance difference among the Optuna samplers, as most exhibit similar behavior, with *GPSampler* showing marginally better results in some cases.

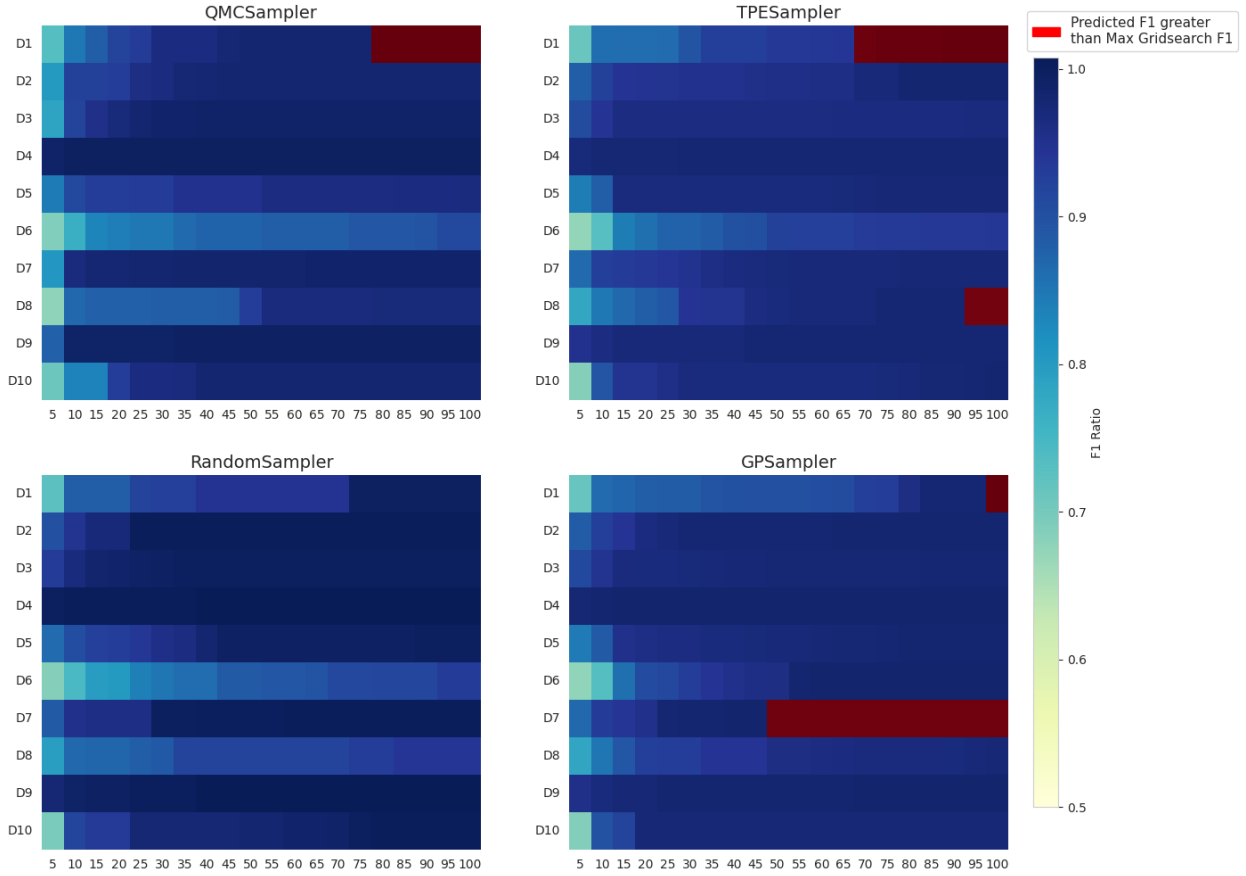

Figure 5.3: Samplers F1-Ratio heatmaps.

To benchmark the Optuna approach, we also conducted an exhaustive grid search for each dataset. The search space, detailed in Table 4.1, explores the Cartesian product of all parameters, resulting in a total of $7 \times 3 \times 19 \times 100 = 39,900$ possible configurations. In contrast, using Optuna, and more generally any hyper-parameter optimization (HPO) framework with available ground-truth, proves to be highly efficient. This is further supported by Figure 5.3, where we observe the F1-Ratio per trial and dataset. In most cases, we begin with an F1-Ratio above 50%, and within 10 to 20 trials, we achieve an F1-Ratio of over 80 to 90%. Notably, in datasets like D1 and D8, samplers achieve even higher F1-scores compared to the exhaustive grid search, which required 39,900 evaluations per dataset. This represents an acceleration in obtaining near-optimal configurations by approximately 99.95%, or in other words, using the ETEER approach allows us to find near-optimal solutions in 99.95% fewer trials than a brute-force solution.

5.3 Experiments with Missing Ground-truth [Problem 2]

As outlined in Section 4.3.3, when the ground-truth is not available, we employed two different approaches. The primary strategy for tackling Problem 2 was to utilize the trials conducted for Problem 1, in conjunction with a set of data features extracted from the

datasets to be matched. We organized the experiments for this task by first dividing the trials from Problem 1 into three distinct categories:

- *Optuna*, which comprises the combined trials from all four Optuna samplers;
- *Gridsearch*, which consists of all trials from the comprehensive grid search benchmarking; and
- *All*, which is the amalgamation of both the *Optuna* and *Gridsearch* trials.

Using these three sets, we trained models on each set following the two learning procedures described in Section 4.3.3. Specifically, we obtained 52,500 trials from each sampler, totaling 5*52,500 = 210,000 trials for the *Optuna* set. The *Gridsearch* set contained 399,000 trials (39,900 trials for each of the 10 datasets). Combining these, the *All* set comprised 609,000 trials. A visualization showing this exact distribution of the training instances can be found in Figure 5.4. For all the training experiments, trials with an F1-score of zero were excluded from the training sets. Additionally, duplicate trials, where samplers proposed configurations already covered by the grid search, were also removed from the training sets.

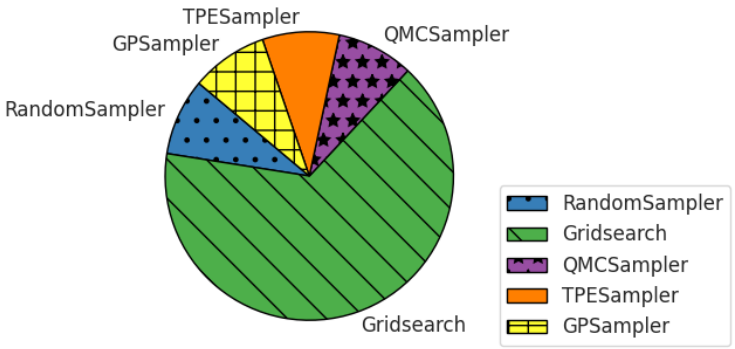

Figure 5.4: All trials training set distribution.

5.3.1 AutoML learning procedure

Using auto-sklearn[4], we conducted experiments by withholding one dataset as the test set while training the AutoML models on all remaining datasets. The results of this approach are summarized in Table 5.2, where we present the best experiments for each dataset. The column labeled *Test set* indicates the dataset used as the test set, while all others served as training sets. The *Trials* column lists the set of trials used for each experiment. *Predicted F1* refers to the actual F1-score of the predicted configuration, measured using pyJedAI. *GB-F1* represents the highest F1-score obtained by any method for each dataset, and *F1-Ratio* measures how close the ETEER prediction was to the best F1-score. See representations 5.1 for the definitions of these terms.

Among all AutoML attempts, implementations of Gradient Boosting and Extra Trees models consistently showed the best performance. Additionally, using the *All* trials set yielded the best results for 6 out of 10 datasets. However, it is also notable that the *Optuna* trials sets produced the best results in 4 out of 10 datasets.

[4] `https://automl.github.io/auto-sklearn`

Table 5.2: Learning procedure 1 (AutoML) results. AutoGB:Gradient Boosting, AutoRF: Ranfom Forest, AutoET:Extra Trees

Test set	Trials	Regressor	Predicted F1	GB-F1	F1-Ratio
D1	optuna	AutoET	54.49	78.43	0.69
D2	all	AutoGB	85.02	85.85	0.99
D3	all	AutoGB	57.42	59.19	0.97
D4	all	AutoRF	98.45	98.60	1.00
D5	optuna	AutoET	75.39	78.92	0.96
D6	optuna	AutoET	50.21	60.42	0.83
D7	all	AutoGB	57.57	67.76	0.85
D8	all	AutoGB	38.09	49.53	0.77
D9	optuna	AutoET	94.39	94.92	0.99
D10	all	AutoGB	55.40	56.12	0.99

The configurations predicted by AutoML achieved near-optimal F1-scores, particularly for datasets D2, D3, D4, D5, D9, and D10, with the minimum *F1-Ratio* being 69% ”close” to the best F1-score. To further illustrate the results of this learning procedure, Figure 5.5 provides a comparative analysis of all datasets across all trial sets. Some variations between the sets of the training trials, can be noticed as they do not exhibit totally similar performance. For example *Gridsearch* in D8, reaches a quite low F1-score. *Optuna* trials dataset seem to have the most robust performance, producing always a high F1-score. As, it will later be presented in Section 5.3.2, employing other algorithmic approaches like, deep learning and Linear Regression, yields better results. For D4, ETEER achieves an optimal score, meaning that it suggests a pipeline, that yields the GB-F1.

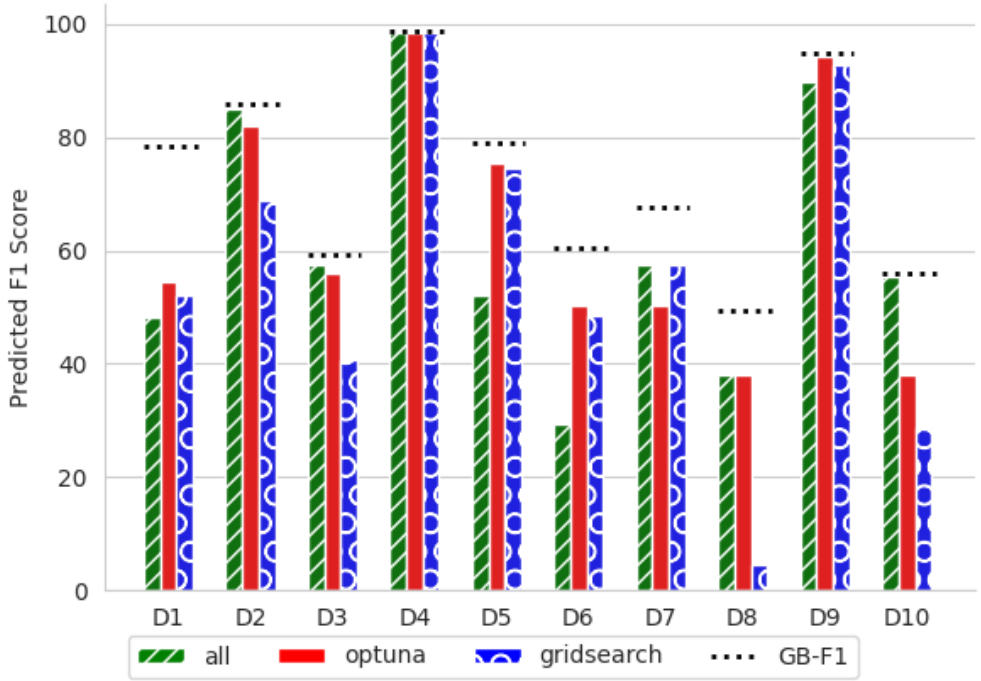

Figure 5.5: AutoML F1-scores.

It is worth noting that the overall runtime and the runtime per model, which are auto-sklearn parameters, are dependent to the resulting performance. More time provided will result probably in an even better model selected. In our experiments, we used the predefined

time limits and did not further investigate the impact of longer runtimes on AutoML's ability to find the optimal model.

5.3.2 Individual regressors learning procedure

In a different manner, we also employed another learning procedure with quite a different methodology. We tested classic regressors provided from sklearn open-source python toolkit, while also created a naive pyTorch[5] Neural Network to test it for this task. A validation set of size 10% from all training data trials is created, and using this we try to minimize the MSE using Optuna as a HPO tool. To better understand this technique please revisit Section 4.3.3 and Learning Procedure 2. The exact configurations tested can be found in Appendix and Table 4.1.

Table 5.3: Learning Procedure 2 results (Individual regressors).

Test set	Trials	Regressor	Predicted F1	GB-F1	F1-Ratio
D1	optuna	RFR	56.77	78.43	0.72
D2	gridsearch	NN	85.40	85.85	0.99
D3	gridsearch	XGB	56.88	59.19	0.96
D4	all	NN	98.51	98.60	1.00
D5	optuna	RFR	76.96	78.92	0.98
D6	optuna	XGB	51.81	60.42	0.86
D7	all	RFR	64.24	67.76	0.95
D8	all	RFR	40.78	49.53	0.82
D9	all	RFR	94.37	94.92	0.99
D10	all	LR	55.80	56.12	0.99

Table 5.3, shows the performance of this learning procedure. Random Forest is shown to be quite effective. Similar with AutoML experiment, *All* training set yields in 5 out of 10 the best results. In a same way like the previous experiment, we provide Figure 5.6, where we notice no big difference between sets of trials. It is now evident that no matter the trials set we will get a similar score, making Optunas trials set, really appealing as it has less than half the size of *All* trials set.

In this scenario, we observe a more balanced distribution in both the performance across different datasets and the utilization of the various training instances (all, optuna, gridsearch). All training sets, as illustrated in Figure 5.4, exhibit comparable performance. However, D1 proves to be the most challenging dataset for the ETEER methodology, as it consistently yields the lowest F1-Ratio.

This technique, compared with AutoML, which encompasses a broader range of algorithmic methods—from simple Linear Regression and Decision Trees to more complex models such as Deep Learning—achieves greater performance and demonstrates resilience to varying data specifications. However, it is more computationally expensive, as it requires conducting a larger number of experiments, unlike AutoML, which offers a more streamlined, one-stop solution.

[5] `https://pytorch.org`

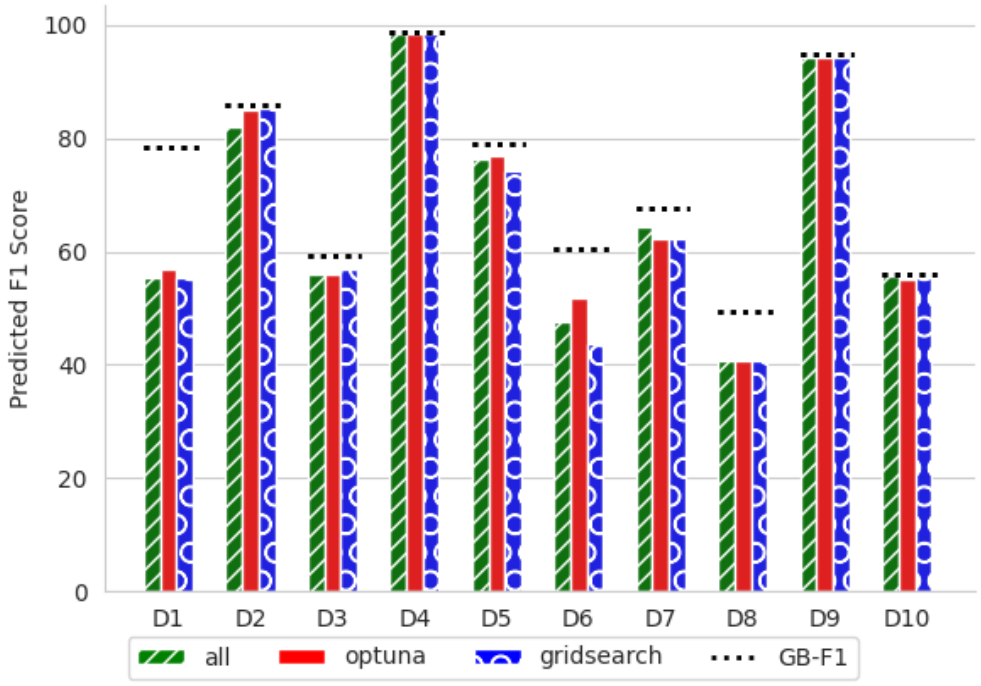

Figure 5.6: Individual regressors F1-scores.

5.3.3 Feature importance

Both AutoML and Individual regressors learning procedures do differentiate the importance of the dataset features. To better understand the predictions made, we did an extensive feature importance analysis based on permutation importance provided be sklearn. For each experiment separately (for every dataset D and for all training trials sets) we calculated feature importance and created an average importance per feature. The results of this analysis are presented in Figure 5.7, which shows the average permutation importance per feature calculated using the AutoML approach, and in Figure 5.8, where we examined the average importance per feature generated by all sklearn regressors, excluding the Neural Network.

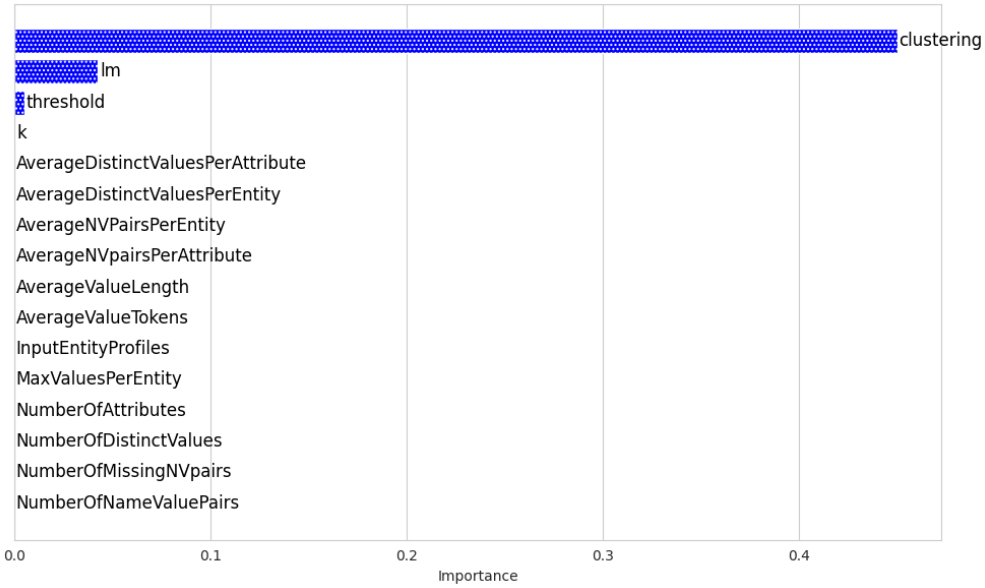

Figure 5.7: AutoML feature importance.

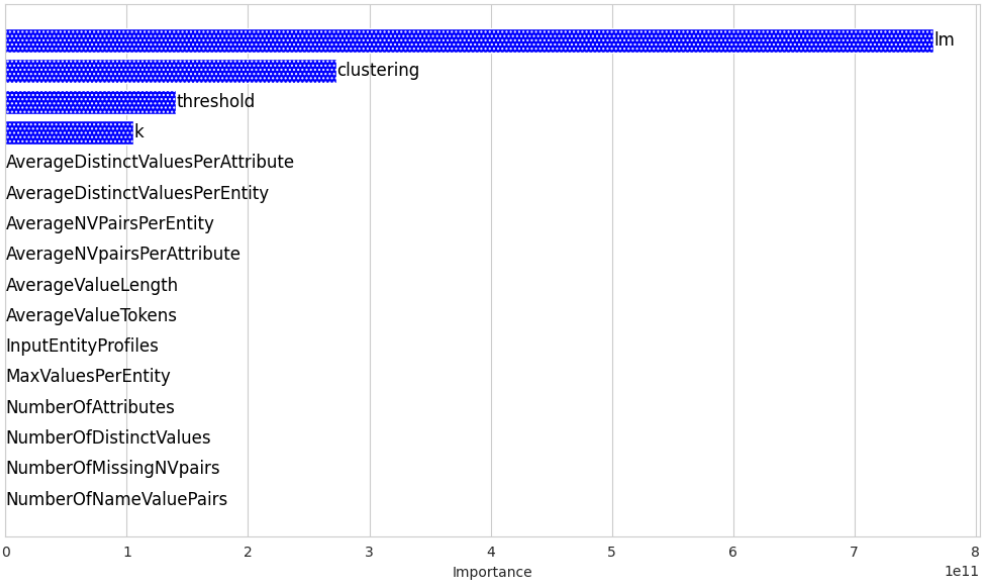

Figure 5.8: Individual regressors feature importance.

The findings of this study can be summarized in the below remarks:

- **[Dataset features]** described in Section 4.3.1 have an importance score of zero. This indicates that their impact on training and predictions is weak or even non-existent. This result is theoretically expected, as the training instances have a large distribution across the four main ETEER parameters, while each dataset feature has only 10 distinct values. With a total of 609,000 training instances, each feature described in Section 4.3.1 takes only 10 distinct values. Consequently, decision tree-based algorithms are unlikely to split based on these features, and linear regression assigns a weight close to zero to them, as they do not provide useful information. While this observation will not be explored further in this thesis, it should be addressed in future iterations of the ETEER methodology.

- **[Parameter: LM]** is the most impactful parameter in the individual regressors approach and the second most impactful in AutoML. This is due to the fact that changing the embedding representation between models like BERT and SentenceBERT results in significantly different entity representations.

- **[Parameter: Clustering]** is the most impactful parameter in the AutoML approach and the second most impactful in individual regressors. This is because clustering is a critical step in the ETEER process, and changing the clustering algorithm leads to a different final step and the application of a totally different algorithm.

- **[Parameter: Threshold]** also plays a significant role, as setting an extremely high or low threshold can drastically alter the performance of entity resolution pipelines. Setting low threshold means that more entities will be marked as matches, whereas setting it high will probably miss some real matching pairs.

- **[Parameter: K]** is the least important parameter, according to the findings. This is because FAISS identifies the top k most similar candidates per entity, and increasing k has minimal impact, as most entities have only a few true matches. Therefore, increasing k does not meaningfully affect the results.

5.3.4 Scalability test

To test the scalability of the ETEER methodology, we applied it to the DBPedia dataset (D11), as described in Section 5.1. DBPedia is one of the largest and most widely used datasets in the ER research literature. The ETEER learning procedure employed is AutoML, with the same settings as in the previous experiments.

Briefly, we first combined all grid search configurations with the features of the DBPedia dataset, resulting in a test set containing 39,900 possible configurations. Next, we trained the AutoML model using the training instances from all datasets (D1, ..., D10) and more specifically from those produced with Optuna. With this model, we predicted the F1-score for each configuration in the DBPedia test set. From the resulting set, ETEER will suggest the top-N (N=1 in general) configuration having the highest predicted F1-scores and will test it in real settings to measure the actual F1-score the suggested configuration. For robustness purposes, we will show the top five (N=5) configurations with the highest predicted F1-scores. The results are presented in Table 5.4.

Table 5.4: Top-5 ETEER configurations proposed for DBPedia.

TopN	LM	K	Clustering	Threshold	F1	Time (s)	Time (h)
1	st5	3	Kiraly Approximate	0.05	84.84	44,260	12.29
2	st5	4	Kiraly Approximate	0.05	84.85	44,002	12.22
3	st5	2	Kiraly Approximate	0.05	84.86	45,914	12.75
4	st5	5	Kiraly Approximate	0.05	84.84	49,334	13.70
5	st5	6	Kiraly Approximate	0.05	84.84	53,496	14.86

Time in Table 5.4 refers to pyJedAI runtime. Where LM, K, Clustering and Threshold constitute the suggested configuration. F1 is the score measured in DBPedia after executing pyJedAI under this configuration.

The results are remarkably surprising, as we achieved the best F1-score, for DBPedia, known to date. Furthermore, not only did the top-ranked pipeline perform well, but all top five pipelines suggested by ETEER captured the underlying distribution effectively. For instance, a well-chosen clustering algorithm, combined with an appropriate threshold and a robust language model, forms the foundation of a strong pipeline, while the parameter k played a relatively minor role.

6. CONCLUSIONS AND FUTURE WORK

This thesis proposes a solution to an underexplored ER area. Our vision is to create ER pipelines that provide hands-off solutions, beneficial for both experts and novice users. By simply running a black-box ER solution, we have managed to address real-world use-case scenarios (with and without ground truth) and tested our methodologies on a broad range of datasets commonly used in the research community. Through thorough experimentation, we covered the spectrum of HPO algorithms in conjunction with ER and presented an alternative solution.

The goals and contributions of this thesis can be summarized in two key points:

(i) The development of a unified framework that implements the ETEER methodology.

(ii) The hands-off construction and configuration of complex, end-to-end pipelines, which can be built by this framework.

With these two primary directions, our objective has been to democratize ER solutions by leveraging open-source ER tools and reducing the complexity of using such tools.

Future Work. There are several potential improvements to the ETEER methodology. First, addressing the feature importance limitations in our approach would be a significant enhancement. Additionally, a major future advancement would involve integrating increasingly complex pipelines. The pyJedAI toolkit, which was used in this thesis, supports two fundamentally different workflows: the Blocking workflow and the Similarity Joins workflow. ETEER should explore these diverse workflows to offer solutions for a broader range of data variations. Finally, the development of a demo application, serving as a component to pyJedAI, would be a valuable addition. This application would accept an entity collection—whether Clean-Clean or Dirty ER—and return an appropriate ER pipeline, aligning with the ultimate goal of this thesis.

ABBREVIATIONS - ACRONYMS

AI	Artificial Intelligence
AO	Architecture Optimization
AutoML	Automated Machine Learning
CCER	Clean-Clean ER
DER	Dirty ER
ER	Entity Resolution
ETEER	End-To-End Entity Resolution
GB	Gradient Boosting
HPO	Hyper-Parameter Optimization
KNN	K-Nearest Neighbor
LM	Language Model
LLM	Large Language Model
LR	Linear Regressor
ML	Machine Learning
NN	Neural Network
QMC	Quasi-Monte Carlo
RFR	Random Forest Regressor
SVM	Support Vector Machine

APPENDIX

Table 6.2: Parameters Tested for Each Regression Method.

Regressor	Parameter	Search Space
Lasso	alpha	LogUniform(10^{-4}, 10^{1})
Ridge	alpha	LogUniform(10^{-4}, 10^{1})
LR	-	-
XGBR	n_estimators	$\{100, ..., 1000\} \subseteq \mathbb{Z}$
	max_depth	$\{3, ..., 10\} \subseteq \mathbb{Z}$
	learning_rate	LogUniform(10^{-3}, 10^{-1})
	subsample	Uniform(0.5, 1.0)
	colsample_bytree	Uniform(0.5, 1.0)
	gamma	$\{0, ..., 5\} \subseteq \mathbb{Z}$
	reg_alpha	LogUniform(10^{-2}, 10^{2})
	reg_lambda	LogUniform(10^{-2}, 10^{2})
RFR	n_estimators	$\{100, ..., 1000\} \subseteq \mathbb{Z}$
	max_depth	$\{3, ..., 10\} \subseteq \mathbb{Z}$
	min_samples_split	$\{2, ..., 20\} \subseteq \mathbb{Z}$
	min_samples_leaf	$\{1, ..., 20\} \subseteq \mathbb{Z}$
	max_features	$\{sqrt, log2\}$
NN	hidden_dim	$\{16, ..., 128\} \subseteq \mathbb{Z}$
	lr	LogUniform($10^{-5}, 10^{-2}$)
	num_epochs	$\{2, ..., 50\} \subseteq \mathbb{Z}$

BIBLIOGRAPHY

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *SIGKDD*, pages 2623–2631. ACM, 2019.

[2] Hotham Altwaijry et al. Query: A framework for integrating entity resolution with query processing. *PVLDB*, 2015.

[3] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, 2015.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.*, 87, 2020.

[5] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search, 2012.

[6] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *TProc. of the 30th International Conference on Machine Learning (ICML 2013), June 2013, pp. I-115 to I-23.*, 2013.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[9] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[10] Christoph Böhm et al. LINDA: distributed web-of-data-scale entity matching. In *CIKM*, pages 2104–2108, 2012.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[12] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(1), mar 2024.

[13] Peter Christen. *Data Matching*. Springer, 2012.

[14] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.*, 53(6):127:1–127:42, 2021.

[15] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[18] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *CoRR*, abs/2401.08281, 2024.

[19] Vasilis Efthymiou, Ekaterini Ioannou, Manos Karvounis, Manolis Koubarakis, Jakub Maciejewski, Konstantinos Nikoletos, George Papadakis, Dimitrios Skoutas, Yannis Velegrakis, and Alexandros Zeakis. Self-configured entity resolution with pyjedai. In Jingrui He, Themis Palpanas, Xiaohua Hu, Alfredo Cuzzocrea, Dejing Dou, Dominik Slezak, Wei Wang, Aleksandra Gruca, Jerry Chun-Wei Lin, and Rakesh Agrawal, editors, *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*, pages 339–343. IEEE, 2023.

[20] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: a survey. *J. Mach. Learn. Res.*, 20(1):1997–2017, jan 2019.

[21] Nikolaos Fanourakis, Vasilis Efthymiou, Dimitris Kotzinos, and Vassilis Christophides. Knowledge graph embedding methods for entity alignment: experimental review. *Data Min. Knowl. Discov.*, 37(5):2070–2137, 2023.

[22] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

[23] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1487–1495. ACM, 2017.

[24] Behzad Golshan, Alon Halevy, George Mihaila, and Wang-Chiew Tan. Data integration: After the teenage years. In *PODS*, pages 101–106, 2017.

[25] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, 2014.

[26] Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, Alexander Statnikov, Sébastien Treguer, and Evelyne Viegas. A brief review of the chalearn automl challenge: Any-time any-dataset learning without human intervention. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 21–30, New York, New York, USA, 24 Jun 2016. PMLR.

[27] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. Framework for evaluating clustering algorithms in duplicate detection. *Proc. VLDB Endow.*, 2(1):1282–1293, 2009.

[28] Ekaterini Ioannou and Minos Garofalakis. Query analytics over probabilistic databases with unmerged duplicates. *TKDE*, 2015.

[29] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[30] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547, 2021.

[31] Patrick Koch, Oleg Golovidov, Steven Gardner, Brett Wujek, Joshua Griffin, and Yan Xu. Autotune: A derivative-free optimization framework for hyperparameter tuning. In *SIGKDD*, pages 443–452, 2018.

[32] Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient deduplication with hadoop. *PVLDB*, 5(12):1878–1881, 2012.

[33] Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *ICDE*, pages 618–629, 2012.

[34] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1):484–493, 2010.

[35] Lars Kotthoff, Chris J. Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *J. Mach. Learn. Res.*, 18:25:1–25:5, 2017.

[36] Simon Lacoste-Julien et al. Sigma: simple greedy matching for aligning large knowledge bases. In *KDD*, pages 572–580, 2013.

[37] Juanzi Li et al. Rimom: A dynamic multistrategy ontology alignment framework. *TKDE*, 21(8):1218–1232, 2009.

[38] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.*, 14(1):50–60, 2020.

[39] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23:1–9, 2022.

[40] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[41] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[42] Sidharth Mudgal et al. Deep learning for entity matching: A design space exploration. In *SIGMOD*, pages 19–34, 2018.

[43] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 19–34, New York, NY, USA, 2018. Association for Computing Machinery.

[44] Konstantinos Nikoletos, Ekaterini Ioannou, and George Papadakis. The five generations of entity resolution on web data. In Kostas Stefanidis, Kari Systä, Maristella Matera, Sebastian Heil, Haridimos Kondylakis, and Elisa Quintarelli, editors, *Web Engineering*, pages 469–473, Cham, 2024. Springer Nature Switzerland.

[45] Konstantinos Nikoletos, George Papadakis, and Manolis Koubarakis. pyJedAI: a lightsaber for Link Discovery. In *Demo at International Semantic Web Conference.*, ISWC, 2022.

[46] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. Embedding-assisted entity resolution for knowledge graphs. In *ESWC*, volume 2873, 2021.

[47] Randal S. Olson and Jason H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *AutoML@ICML*, 2016.

[48] Matteo Paganelli, Francesco Del Buono, Francesco Guerra, Marco Pevarello, and Maurizio Vincini. Automated machine learning for entity matching tasks. In *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*, Short Papers, pages 325–330. OpenProceedings.org, 2021.

[49] George Papadakis, Vasilis Efthymiou, Emmanouil Thanos, Oktie Hassanzadeh, and Peter Christen. An analysis of one-to-one matching algorithms for entity resolution. *VLDB J.*, 32(6):1369–1400, 2023.

[50] George Papadakis et al. Three-dimensional entity resolution with jedai. *Inf. Syst.*, 93, 2020.

[51] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2):31:1–31:42, 2021.

[52] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *PVLDB*, 9(9), 2016.

[53] Marios Papamichalopoulos et al. Three-dimensional geospatial interlinking with jedai-spatial. *CoRR*, abs/2205.01905, 2022.

[54] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, 2014.

[55] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*, 2019.

[56] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP*, pages 3980–3990, 2019.

[57] Alieh Saeedi, Eric Peukert, and Erhard Rahm. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *ADBIS*, volume 10509, pages 278–293, 2017.

[58] Alieh Saeedi, Eric Peukert, and Erhard Rahm. Incremental multi-source entity resolution for knowledge graph completion. In *ESWC*, volume 12123, pages 393–408, 2020.

[59] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[60] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[61] Xingyou Song, Sagi Perel, Chansoo Lee, Greg Kochanski, and Daniel Golovin. Open source vizier: Distributed infrastructure and api for reliable and flexible black-box optimization. In *Automated Machine Learning Conference, Systems Track (AutoML-Conf Systems)*, 2022.

[62] Fabian M. Suchanek et al. PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[63] Zequn Sun, Qingheng Zhang, Wei Hu, Chengming Wang, Muhao Chen, Farahnaz Akrami, and Chengkai Li. A benchmarking study of embedding-based entity alignment for knowledge graphs. *Proc. VLDB Endow.*, 13(11):2326–2340, 2020.

[64] Saravanan Thirumuruganathan et al. Deep learning for blocking in entity matching: A design space exploration. *PVLDB*, 14(11):2459–2472, 2021.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[66] Pei Wang, Weiling Zheng, Jiannan Wang, and Jian Pei. Automating entity matching model development. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1296–1307, 2021.

[67] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[68] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. Zeroer: Entity resolution using zero labeled examples. In *SIGMOD*, pages 1149–1164, 2020.

[69] Alexandros Zeakis, George Papadakis, Dimitrios Skoutas, and Manolis Koubarakis. Pre-trained embeddings for entity resolution: An experimental analysis. *Proc. VLDB Endow.*, 16(9):2225–2238, 2023.

[70] Chen Zhao and Yeye He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*, WWW '19, page 2413–2424, New York, NY, USA, 2019. Association for Computing Machinery.