A Deep Dive Into Cross-Dataset Entity Matching with Large and Small Language Models

Zeyu Zhang
University of Amsterdam & Amsterdam UMC
z.zhang2@uva.nl

Paul Groth
University of Amsterdam
p.t.groth@uva.nl

Iacer Calixto
Amsterdam UMC, University of Amsterdam
i.coimbra@amsterdamumc.nl

Sebastian Schelter
BIFOLD & TU Berlin
schelter@tu-berlin.de

ABSTRACT

Entity matching (EM) is the problem of determining whether two data records refer to the same real-world entity. A particularly challenging scenario is *cross-dataset entity matching*, where the matcher has to work with an unseen target dataset for which no labelled examples are available. Cross-dataset EM is crucial in scenarios where a high level of automation is required, and where it is unlikely or impractical to force a domain expert to manually label training data. Recently, approaches based on language models have become popular for EM, and often promise impressive transfer capabilities. However, there is a lack of a comprehensive and systematic study of the cross-dataset EM capabilities of these recent approaches. It is unclear, which categories of language models are actually applicable in a cross-dataset EM setting, how well current EM approaches perform when they are evaluated systematically under a cross-dataset setting, and what the relationship between the prediction quality and deployment cost of various large language model-based EM approaches is.

We address these open questions with the first comprehensive and systematic study on cross-dataset entity matching, where we evaluate eight matchers on 11 benchmark datasets, cover a wide variety of model sizes and transfer learning approaches, and also explore and quantify the relation between prediction quality and deployment cost of the matching approaches. We find that fine-tuned small models can perform on par with prompted large models, that data-centric approaches outperform model-centric approaches and that approaches using well-performing small models can be deployed at an orders of magnitude lower cost than comparably performing approaches with large commercial models.

1 INTRODUCTION

Entity matching (EM) is the problem of determining whether two data records refer to the same real-world entity. EM is a well-studied problem over the past decade [5, 7, 12, 14, 17, 31, 36, 38, 52] due to its high practical importance in data integration [2, 18, 22, 48, 61].

The need for cross-dataset entity matching. A typical restriction in entity matching tasks is the reliance on labelled examples, which are often scarce due to the high cost of human annotation. A less restrictive yet more challenging scenario is *cross-dataset entity matching* [50, 55, 60], where the matcher has to work with an unseen target dataset for which no labelled examples are available. The cross-dataset EM setup is crucial in cloud scenarios where a high level of automation required, and where it is unlikely or impractical to force a domain expert to manually label training data. According to a recent talk from Databricks for example, their top customers maintain hundreds of thousands of tables on average [58] and all operations on those tables must be fully automated. Moreover, data integration services such as AWS Glue [45] currently still require end users to manually label examples for entity matching [46], which limits their applicability. Additionally, cross-dataset matchers can be applied for duplicate detection [23] as part of data cleaning in machine learning pipelines [1, 30], and are also valuable as a primitive for entity alignment in tasks such as table reclamation [15].

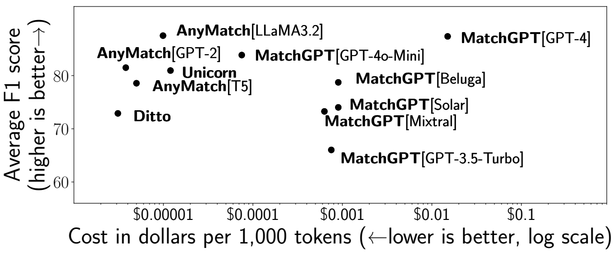

Figure 1: Prediction quality vs. deployment cost for language model-based EM approaches in a cross-dataset setting.

Open questions. Numerous EM algorithms and models have been proposed in the past; however, few studies have explored them in the cross-dataset context. Moreover, existing research often focuses solely on comparing the predictive quality of different methods, but overlooks the computational costs associated with these models. These shortcomings lead us to the following three research questions:

RQ1: Which categories of language models are actually applicable in a cross-dataset EM setting?

There exists a large body of research on the EM task. Classical rule-based approaches rely heavily on schema information and semantic relationships to craft matching rules. Machine learning (ML)-based approaches, on the other hand, involve building a model to capture the similarity between entities. The Magellan system [12] leverages classical machine learning methods. In recent years, several deep learning-based solutions have been proposed [36], for example based on graph neural networks [7, 17] or transformers [56, 59]. However, these approaches are not tailored to the cross-dataset setting. The advent of language models has led to a resurgence of research on EM, existing

10.48786/edbt.2025.75

approaches either fine-tune small language models (SLMs) [31, 50, 62] or leverage large language models (LLMs) for EM via prompting [29, 37, 41, 60]. Some methods are explicitly designed to address the cross-dataset setting as demonstrated by `ZeroER` [55] and `AnyMatch` [62]. But in general, a careful inspection of the model design and assumptions about the input data is required to decide whether a given approach can even be deployed in real-world cross-dataset settings.

RQ2: *How well do current EM approaches perform when evaluated systematically under a cross-dataset setting?*

The majority of EM approaches are not specifically designed for the cross-dataset setting and their cross-dataset capabilities are often only evaluated on a small number of arbitrarily chosen held-out datasets. For example, `Jellyfish` [60] is only evaluated on two datasets with e-commerce products, while `Unicorn` [50] is only evaluated on a single target dataset containing pairs of academic publications, and even the comprehensive evaluation of `MatchGPT` [41] still lacks common datasets from the food and music domain. Furthermore, these studies often assume an independent distribution of data samples across datasets, and neglect the question of how domain similarity between datasets influences performance. This makes it difficult to judge the overall performance under a cross-dataset setting and necessitates a systematic evaluation on a larger number of datasets.

RQ3: *What is the trade-off between the prediction quality and deployment cost of various language model-based EM approaches?*

In recent years, language models have been widely applied to structured data, for example to generate SQL queries from natural language [6, 40] or for data preparation [28, 51, 54]. While there is a lot of enthusiasm for leveraging language models for entity matching, several of the proposed approaches rely on extremely large proprietary models with hundreds of billions or even trillions of parameters [53], which require expensive accelerator hardware for deployment. As a result, these approaches incur a hefty cost at a low throughput during inference. The latest commercial LLMs often come with a *"throughput of less than 1 KB per second"* [32] at a high cost imposed by their pay-per-token model, which results in prices of *"5 USD for processing just 5MB of data"* [32]. LLM-based matchers inherit this high computational cost, which severely limits their scalability and applicability. An indication of this is the fact that even relatively small-scale academic datasets are down-sampled for experimentation with LLM-based approaches [41]. We argue that this cost needs to be taken into account, especially with respect to large-scale cloud deployments of such models.

Overview. We address the outlined research questions with an extensive experimental study. In total, we use over 425 GPU hours and spend more than 290 dollars on OpenAI API calls for our experiments. For RQ1, we list and categorise existing entity matchers with cross-dataset capabilities in Section 3. We begin by comparing matchers that are either parameter-free or do not require explicit fine-tuning. For matchers leveraging pretrained language models (PLMs), we categorise them into two groups: model-aware and model-agnostic matchers. The former require a carefully designed model structure, while the latter re-use an off-the-shelf model and focus on the fine-tuning data only.

Next, we discuss our evaluation methodology in Section 2, where we formalise the cross-dataset EM task, describe how to systematically evaluate the cross-dataset performance of EM approaches with a "leave-one-dataset-out" methodology, and lay out how we estimate the deployment cost of both openly available and proprietary LLM-based approaches. Based on this foundation, we tackle RQ2 and RQ3 in Section 4, where we evaluate both the matching performance and computational cost of the outlined methods. We compare the performance and cost differences of approaches that leverage prompted LLMs with matchers fine-tuned on PLMs through detailed experimental analysis. We additionally investigate the benefits of providing additional demonstrations when prompting LLMs, using several popular, state-of-the-art, open-source LLMs in a cross-dataset setting. Finally, we summarise the lessons learned and limitations from our study in Section 5, and outline future research directions.

Contributions. In summary, this paper provides the following contributions:

- *Comprehensive study* – We conduct the first comprehensive and systematic study on cross-dataset entity matching, where we evaluate eight matchers on 11 benchmark datasets, and cover a wide variety of model sizes, domain specializations, and transfer learning approaches (Sections 2, 3 & 4.1).
- *Consideration of deployment cost* – We explore and quantify the trade-off between prediction quality and deployment cost of the matching approaches (Section 4.2).
- *Findings* – We discuss our findings, e.g., that fine-tuned small models perform on par with prompted large models, that data-centric approaches outperform model-centric approaches and that approaches using well-performing small models can be deployed at an orders of magnitude lower cost than comparably performing approaches with large commercial models (Section 5).
- *Reproducibility* – We organize the source code of all available baselines within a unified evaluation framework, ensuring that they are compatible and directly comparable. Our codebase is publicly released at: https://github.com/Jantory/cross-dataset-em-study

2 EVALUATION FRAMEWORK

We discuss our study design in the following. We first formally define the cross-dataset EM setup (Section 2.1), then detail our systematic evaluation via a "leave-one-dataset-out" strategy (Section 2.2), and finally discuss how we estimate the deployment cost of various LLM-based approaches (Section 2.3).

2.1 Cross-Dataset Entity Matching

Entity matching. The entity matching problem is to predict whether the pair of records (r_l, r_r) with $r_l \in \mathrm{R}_{\mathrm{left}}$ and $r_r \in \mathrm{R}_{\mathrm{right}}$ refers to the same real-world entity or not. $\mathrm{R}_{\mathrm{left}}$ and $\mathrm{R}_{\mathrm{right}}$ denote two input relations with k aligned attributes $A = \{a_1, \ldots, a_k\}$.

Entity matching is often modelled as a binary classification problem with a labelled training set $\mathcal{D}_{\mathrm{train}} \subset \mathrm{R}_{\mathrm{left}} \times \mathrm{R}_{\mathrm{right}} \times \{0, 1\}$. State-of-the-art approaches [31] featurise the example pairs based on their attribute names $a_1, \ldots, a_k$ and the aligned attribute values $v_{l_1} = r_l[a_1], \ldots, v_{l_k} = r_l[a_k], v_{r_1} = r_r[a_1], \ldots, v_{r_k} = r_r[a_k]$. Furthermore, the attribute values may be augmented with additional data, e.g., domain information. Note that our setup assumes that there are no duplicates within a single relation. Furthermore, real-world entity matching systems typically first apply a blocking function to the set $R_l \times R_r$ to form smaller candidate sets as input to the matcher. In this study, we focus on the matchers themselves and evaluate general approaches which can be easily plugged into existing matching systems.

Cross-dataset entity matching. In contrast to classical entity matching, we are interested in a more challenging setting, referred to as *cross-dataset entity matching*. In particular, we investigate the entity matching problem under the following two restrictions:

Restriction 1 - Unseen target data: *A cross-dataset matcher will not have access to labelled example pairs for the target relations* R_{left} *and* R_{right}*, which means there is no training set available for the unseen target data* $\mathcal{D}_{target}$.

Restriction 2 - Lack of type information: *There is no column name or column type information accessible for the target relations* R_{left} *and* R_{right}*. A cross-dataset matcher can only enumerate the attribute values* $r[a_1], \ldots, r[a_k]$ *of a record* r *from the target relations in a string representation.*

Use cases. A cross-dataset EM setup is crucial in scenarios where a high level of automation required, and where it is unlikely or impractical to force a domain expert to manually label training data. Furthermore, schema and type information may often be missing or unreliable in these scenarios. For instance, columns in real-world tables are often wrongly typed as string columns and have cryptic, hard-to-interpret names [13, 33].

Examples for such use cases include data integration services in the cloud, such as AWS Glue [45], which provide automated integration capabilities on enterprise data lakes. Currently, such services require end users to manually label examples for entity matching [46]. They match our restrictions well as they need to be able to ingest and automatically process heterogeneous data from various data sources (e.g., data in relational databases or Excel and CSV files from distributed file systems). Providing competitive matching performance out-of-the-box without labelled examples would greatly improve the applicability of these services.

Another use case is duplicate detection and removal in ML pipelines [1, 2, 23, 30], where the input dataset often originates from various non-relational sources such log files and rarely comes with labelled examples for potential duplicates. Large organisations run hundreds of such pipelines in production [57]. Furthermore, cross-dataset entity matching is also valuable as a primitive for entity alignment in other data integration tasks. An example is table reclamation [15], where a "fast, approximate instance comparison algorithm" is required for future work. Note that these use cases depend on cost-efficient matchers and may often require scalable matching on hundreds of thousands or even millions of records.

2.2 "Leave-One-Dataset-Out" Evaluation

In the following, we introduce our evaluation strategy, starting with a description of the dataset and the synthesis process employed to address cross-dataset constraints. We then outline the evaluation metrics used to assess the model's performance, considering both quality and cost dimensions.

Datasets. We experiment on 11 benchmark datasets detailed in Table 1, including those commonly used in entity matching research [31, 37, 41] and expand upon them. Their sample sizes range from hundreds to tens of thousands. Additionally, they span a variety of domains and include varying numbers of attributes. To prevent data leakage during the fine-tuning phase, we conduct a comprehensive comparison across different dataset pairs and validate that no data samples are shared between datasets.

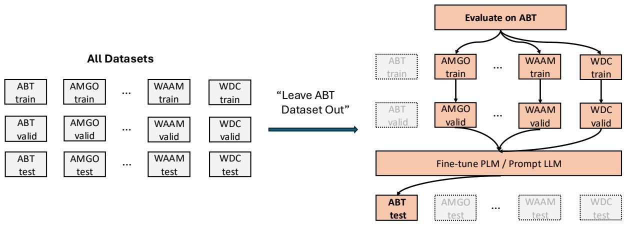

Figure 2: Example evaluation on the ABT dataset using the "leave-one-dataset-out" strategy.

Methodology. Evaluating models for the cross-dataset setting is challenging, since example data for transfer learning is required. Cross-dataset capabilities of entity matchers have not been systematically evaluated in previous studies. When considered, evaluations are typically conducted on a small number of arbitrarily chosen held-out datasets. For instance, Jellyfish [60] and Unicorn [50] are universal solutions for table-relevant tasks that include cross-dataset EM support. However, despite the availability of full benchmark datasets, Jellyfish evaluates model performance on only two e-commerce product datasets, while Unicorn is assessed on a single dataset comprising pairs of academic publications. Even the comprehensive evaluation of MatchGPT [41] relies on only six datasets, which limits its coverage across diverse domains. In contrast, our goal is to conduct a fair and thorough evaluation, rather than selecting only a single dataset combination as seen in previous studies.

Table 1: The 11 benchmark datasets from [10, 41], organized by corresponding domain along with key statistics.

	Dataset	Domain	#Attr.	#Pos.	#Neg.
ABT	Abt-Buy	web product	3	1,028	8,547
WDC	Web Data Commons	web product	3	2,250	7,992
DBAC	DBLP-ACM	citation	4	2,220	10,143
DBGO	DBLP-Google	citation	4	5,347	23,360
FOZA	Fodors-Zagats	restaurant	6	110	836
ZOYE	Zomato-Yelp	restaurant	7	90	354
AMGO	Amazon-Google	software	3	1,167	10,293
BEER	Beer	drink	4	68	382
ITAM	iTunes-Amazon	music	8	132	407
ROIM	RottenTomato-IMDB	movie	5	190	410
WAAM	Walmart-Amazon	electronics	5	962	9,280

Since there is no established benchmark for evaluating the cross-dataset capabilities of entity matchers, we adopt a strategy we term as 'leave-one-dataset-out' evaluation to address the cross-dataset constraints: to test on a given unseen target dataset, we allow the matcher to access the other ten datasets as transfer learning data. These ten datasets can be utilised to construct the data for fine-tuning the matching model or to select demonstrations for prompting the language model e.g., when evaluated on ABT, a model can use the remaining ten datasets (WDC, DBAC, DBGO, FOZA, ZOYE, AMGO, BEER, ITAM, ROIM, and WAAM) for transfer learning, and can optionally select demonstrations from them for prompting. A visual illustration of this process can be found in Figure 2.

Note that this methodology adheres to the cross-dataset restrictions in Section 2.1, as no examples from the target dataset are used during fine-tuning or prompting, and we ensure that no column names are used as well. We make all variants of the trained models available as part of our supplemental material.

Metrics. In line with existing research, we report the F1 score for each setting, which combines precision and recall , and is particularly useful in situations where the class distribution is imbalanced. Formally, the F1 score is defined as:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

and

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

To compare the predictive quality of different baselines, we calculate the macro-averaged F1 score for each dataset, treating all datasets as equally important. This is represented by the "Mean" column in Table 2 and Table 3.

Repetitions. To quantify the uncertainty of different runs, we employ five distinct random seeds to conduct experiments multiple times. When a language model is used as the matcher, data serialization is always required, converting records from relations into strings. As language models are sensitive to the input sequence, we also use the corresponding random seeds to vary the serialised input for the language model. Specifically, during serialization, we randomly shuffle the order of columns to alter their occurrence in the input sequence. These different sequences are input to the same model to generate predictions. We report both the mean F1 score and its standard deviation when presenting the results.

2.3 Measuring the Quality-Cost Trade-off

In addition to evaluating the predictive performance of various language models, we also compare their associated computational costs in terms of dollars per 1K tokens processed. The models employed for this comparison fall into two categories: publicly available models and API-based proprietary models. For the publicly available models, we estimate their inference throughput and calculate their deployment cost based on the GPU machine rates from cloud from cloud vendors such as Amazon Web Services. In contrast, for the proprietary API-based models, we directly use the provider's listed cost per 1K tokens for comparison. Since all approaches frame entity matching under a language model scenario as a sentence classification task, the model is required to generate only a binary output—either "Yes" or "No"—for a given input. As the input length is significantly longer than the output, we disregard the output cost in our evaluation and focus solely on the cost of processing the input.

By systematically comparing both types of models in terms of performance and cost, we give insights into the trade-off involved in choosing between publicly available and proprietary solutions for entity matching tasks. Ultimately, this analysis not only contributes to the understanding of model efficiency but also assists practitioners in making informed decisions based on their specific resource constraints and application needs.

3 CROSS-DATASET EM APPROACHES

In this section, we address RQ1 by outlining matching approaches with cross-dataset capabilities, which we include in our study (an overview can be found in Table 2). Most of these are based on pretrained language models, with one exception being a parameter-free approach. Following a recent survey [16], we classify a language model as a large language model if its parameter size exceeds one billion, while models with fewer parameters are categorised as small language models. We refer to approaches which require custom model architectures as model-aware matchers, and those requiring no model customisation as model-agnostic matchers.

Table 2: An overview of the methods with cross-dataset capabilities included in our study.

Matcher	PLM	Type	Availability
`ZeroER`	No	Parameter-free	https://github.com/chu-data-lab/zeroer
`Ditto`	Small	Model-aware	https://github.com/megagonlabs/ditto
`Unicorn`	Small	Model-aware	https://github.com/ruc-datalab/Unicorn
`AnyMatch`	Small	Model-agnostic	https://github.com/Jantory/anymatch
`Jellyfish`	Large	Model-agnostic	https://huggingface.co/NECOUDBFM/Jellyfish-13B
`TableGPT`	Large	Model-agnostic	https://github.com/microsoft/Table-GPT
`MatchGPT`	Large	Model-agnostic	https://github.com/wbsg-uni-mannheim/MatchGPT

3.1 Parameter-Free Models

ZeroER. A seminal work in the cross-dataset entity matching area is *ZeroER* [55], which is a parameter-free method, explicitly designed for the cross-dataset case without any training data for the target dataset (termed as "zero-shot" EM in the original work). The approach is built on the observation that the similarity vectors for matching records are distributed differently than the similarity vectors for non-matching records. However, this approach has several drawbacks: it requires information about the column types and the selection of similarity functions, is only applicable in a batch setting (making it unable to match individual record pairs in isolation, which complicates debugging false predictions), and relies on distributional assumptions that may not hold for every dataset.

3.2 Fine-Tuned Small Language Models

Ditto. The state-of-the-art matcher *Ditto* [31] is based on fine-tuning an encoder language model with a separate prediction head. To ensure domain specificity, Ditto incorporates domain knowledge during data serialisation and augments the training data to enhance the model's ability to distinguish between challenging entity pairs. This process includes actions such as dropping columns and editing spans of tokens.

Although Ditto is not specifically designed for cross-dataset EM, it does not rely on a hard-coded schema during training, making it applicable to unseen target datasets with a different schema.

Unicorn. The unified multi-tasking model named *Unicorn* [50] is designed to support seven matching tasks in data integration, including entity matching. Unicorn applies multi-task learning by using a mixture of expert architecture [34], aiming to learn specialised expert models for different tasks. It firstly generates data samples from different tasks to be serialised with their respective schema and encoded using a pretrained language model. Then, task-specific expert models transform these representations into task-specific embeddings. Finally, these embeddings are merged and fed into a matching module. Through the use of multi-task experts, Unicorn enables the model to learn distinct embeddings for different tasks, which allows it to generalise and perform well on unseen datasets and tasks.

Ditto and Unicorn both leverage encoder-only [44] language models to encode serialised inputs into representations, and require a customisation of the model architecture. Despite being straightforward, this approach introduces complexity to the matcher design as a careful construction of the prediction head is required, which subsequently impacts the model's quality. Additionally, adapting to new model releases may necessitate a redesign, as the dimensions of the representations may not always remain consistent.

In recent years, we have witnessed the success of other types of transformer-based models, referred as encoder-decoder and decoder-only models [44]. These models do not require the extraction of intermediate representations and can directly generate sequential outputs that are easily interpretable by humans. Fine-tuning these models is simple, as the model structure remains intact, and only input-output sequence pairs need to be prepared.

AnyMatch. The recently proposed *AnyMatch [62]* serve as a model-agnostic, data-centric framework to fine-tune encoder-decoder and decoder-only language models. AnyMatch does not modify the model structure; instead, it emphasizes generating high-value fine-tuning data via several data selection techniques such as boosting to identify difficult examples and offers heuristics to balance the label distribution, and optionally supports augmenting the fine-tuning data with attribute-level examples. The cross-dataset EM capability is derived from the model's contextual understanding, enabling it to make predictions to unseen domain data.

3.3 Instruction-Tuned or Continual-Tuned Language Models

Another line of research applicable to cross-dataset EM focuses on enhancing models' general understanding of tabular contexts through specific instruction tuning or continued pre-training.

Jellyfish. The general LLM-based approach *Jellyfish* [60] targets four data preprocessing tasks (including entity matching). Jellyfish leverages two LLaMA2-13B models fine-tuned in an instruction tuning fashion. The corresponding tuning data is specifically created to accommodate multiple data preprocessing tasks. Essentially, one LLaMA model is tasked with classification, providing detailed reasoning, while the second model interprets this output to refine the reasoning process further. Jellyfish is explicitly designed to address out-of-domain data preparation scenarios on unseen datasets, making it well-suited for cross-dataset EM.

TableGPT. The work on *TableGPT* [29] enhances LLMs with a continuous learning setup. This approach utilises diverse table tasks synthesised from real tables as training corpus, and continues the pretraining step of the language models on this data, which improves the tabular understanding. The newly pretrained model can then be prompted for downstream table-relevant tasks. TableGPT can adhere to the cross-dataset EM constraints just like the standard approach of prompting LLMs.

3.4 Prompted Large Language Models

Next, we discuss other model agnostic approaches that rely on prompting LLMs for EM purpose.

GPT-3. Narayan et al. [37] have pioneered this area by prompting the large commercial LLM GPT-3 [4] for three data wrangling tasks, including entity matching. This work systematically analyses how different serialisation methods and demonstration selection can impact the performance of LLMs. Their experimental results showcase that prompting large commercial LLMs with serialised records can achieve competitive matching performance compared to fine-tuned SLMs. They also report that prompting with randomly selected demonstrations from the target dataset can hurt model performance compared to prompting without demonstrations. The prompting strategy for EM without demonstrations can be viewed as a natural cross-dataset matcher. This work also inspires us to evaluate the impact of deliberately selected demonstrations not from the target dataset, which exemplifies the cross-dataset scenario we are addressing.

MatchGPT. The work on *MatchGPT* [41] comprehensively explores the design space of using large language models (LLMs) for entity matching. Two key perspectives are considered in building the system: base model selection and prompt design. They compare the performance of various LLMs, ranging from openly available models to commercial API-based models, introduce several prompting formats and evaluate performance variations across these. They also explore the impact of demonstrations during prompting and do not adhere to cross-dataset constraints. Our study differs from these two studies as follows: we focus on cross-dataset entity matching only, additionally include fine-tuned SLMs, and not only consider on predictive quality but also on the inference cost of the model.

4 EVALUATION & ANALYSIS

Next, we present and discuss the experimental results of our study with respect to both the prediction quality (RQ2, Section 4.1) and deployment cost (RQ3, Section 4.2).

4.1 Prediction Quality

In our first experiment, we measure the prediction quality of the above-mentioned entity matching approaches with cross-dataset capabilities from Section 3.

Data preparation. We experiment on the 11 benchmark datasets detailed in Section 2.2. Due to the high costs of commercial LLM APIs, the `MatchGPT` study down-samples a test set if it exceeds 1,250 samples. We adopt their strategy and include a maximum of 1,250 randomly chosen samples for all test sets, while ensuring that the test sets used for evaluation remain identical across all compared baselines.

Model configurations. We detail the exact configurations for the matchers outlined in section 3.

Parameter-free baselines. We start from a trivial baseline method `StringSim`, which serialises both input tuples to compare by casting each column to a string and concatenating the values with a comma separator. This method computes the string similarity of the serialised tuples via the Ratcliff/Obershelp algorithm from Python's `difflib` package and predicts a match if the corresponding similarity is greater than 0.5. For `ZeroER`, we leverage the implementation provided in the REIN benchmark [1]. To adhere to the cross-dataset restriction 1 from Section 2.1, the model is exposed only to samples in test partitions. However, since `ZeroER` requires knowledge of column types to select appropriate similarity metrics for computation, it partially violates the cross-dataset restriction 2.

Fine-tuned small language models for entity matching. For `Ditto`, we leverage the original code with identical hyperparameters from the authors' GitHub repository and configure the model to use BERT [11] as the base language model. During the experiment, we apply their "data augmentation" and "summarisation" strategies but omit the optimisation of

Table 3: Average F1 scores and standard deviations for cross-dataset entity matching (best in bold, second-best underlined, model seen datasets during training in brackets, and same-domain datasets in same colour). AnyMatch [LLaMA3.2] performs on par with the trillion-parameter GPT-4, which has three orders of magnitude more parameters.

	#params	Unseen Target Dataset											
	(millions)	ABT	WDC	DBAC	DBGO	FOZA	ZOYE	AMGO	BEER	ITAM	ROIM	WAAM	Mean
StringSim	-	$32.2_{\pm 0.0}$	$32.5_{\pm 0.5}$	$73.7_{\pm 0.6}$	$59.8_{\pm 0.6}$	$22.5_{\pm 0.7}$	$45.9_{\pm 1.7}$	$36.9_{\pm 0.2}$	$33.6_{\pm 2.7}$	$50.9_{\pm 0.7}$	$62.7_{\pm 0.8}$	$28.0_{\pm 0.1}$	$43.5_{\pm 0.1}$
ZeroER	-	$37.6_{\pm 0.0}$	$41.2_{\pm 0.0}$	$93.7_{\pm 0.0}$	$59.1_{\pm 0.0}$	$93.9_{\pm 0.0}$	$88.2_{\pm 0.0}$	$23.3_{\pm 0.0}$	$61.9_{\pm 0.0}$	$10.8_{\pm 0.0}$	$79.7_{\pm 0.0}$	$38.7_{\pm 0.0}$	$57.1_{\pm 0.0}$
Ditto	110	$67.8_{\pm 2.6}$	$43.1_{\pm 4.1}$	$94.4_{\pm 0.4}$	$69.7_{\pm 8.2}$	$92.5_{\pm 5.0}$	$78.5_{\pm 13.5}$	$59.4_{\pm 0.9}$	$89.1_{\pm 4.7}$	$65.7_{\pm 7.2}$	$79.1_{\pm 9.8}$	$62.4_{\pm 5.9}$	$72.9_{\pm 2.6}$
Unicorn	143	$87.8_{\pm 2.0}$	$71.9_{\pm 1.4}$	$90.6_{\pm 3.8}$	$86.4_{\pm 2.8}$	$86.8_{\pm 8.1}$	$95.2_{\pm 5.1}$	$64.0_{\pm 3.5}$	$80.2_{\pm 3.8}$	$65.8_{\pm 10.6}$	$90.1_{\pm 4.4}$	$71.9_{\pm 0.8}$	$81.0_{\pm 1.0}$
AnyMatch [GPT-2]	124	$76.5_{\pm 3.8}$	$60.3_{\pm 3.5}$	$95.2_{\pm 0.6}$	$85.7_{\pm 1.0}$	$96.4_{\pm 1.1}$	$95.1_{\pm 4.2}$	$55.9_{\pm 1.3}$	$91.2_{\pm 2.5}$	$\mathbf{85.0}_{\pm 5.8}$	$89.3_{\pm 6.0}$	$66.0_{\pm 5.6}$	$81.5_{\pm 1.4}$
AnyMatch [T5]	220	$76.0_{\pm 4.0}$	$55.4_{\pm 4.6}$	$96.4_{\pm 0.5}$	$75.0_{\pm 6.2}$	$95.4_{\pm 2.1}$	$95.5_{\pm 4.1}$	$64.4_{\pm 3.3}$	$89.2_{\pm 3.7}$	$79.6_{\pm 9.1}$	$72.0_{\pm 11.4}$	$65.5_{\pm 8.1}$	$78.6_{\pm 1.8}$
AnyMatch [LLaMA3.2]	1,300	$\underline{89.3}_{\pm 0.9}$	$69.4_{\pm 2.2}$	$\underline{96.5}_{\pm 0.5}$	$\underline{89.8}_{\pm 1.1}$	$\mathbf{99.6}_{\pm 0.9}$	$\underline{98.2}_{\pm 1.9}$	$69.3_{\pm 2.2}$	$\mathbf{95.3}_{\pm 2.5}$	$\underline{82.3}_{\pm 8.8}$	$95.9_{\pm 1.3}$	$77.2_{\pm 7.0}$	$\mathbf{87.5}_{\pm 1.0}$
Jellyfish	13,000	$79.2_{\pm 2.8}$	$73.0_{\pm 0.6}$	$(\mathbf{97.7}_{\pm 0.6})$	$(\mathbf{93.4}_{\pm 0.6})$	$(\underline{97.3}_{\pm 0.9})$	$\mathbf{99.1}_{\pm 1.2}$	$(\underline{72.1}_{\pm 3.3})$	$(\underline{90.1}_{\pm 5.6})$	$(51.4_{\pm 1.6})$	$\underline{97.0}_{\pm 2.4}$	$81.4_{\pm 3.0}$	$84.7_{\pm 0.7}$
MatchGPT [Mixtral-8x7B]	56,000	$80.7_{\pm 5.3}$	$69.5_{\pm 1.8}$	$92.2_{\pm 3.3}$	$71.4_{\pm 3.4}$	$88.6_{\pm 6.0}$	$91.0_{\pm 5.0}$	$28.1_{\pm 2.2}$	$75.9_{\pm 10.7}$	$53.8_{\pm 6.4}$	$86.0_{\pm 4.7}$	$68.8_{\pm 8.4}$	$73.3_{\pm 0.9}$
MatchGPT [SOLAR]	70,000	$76.4_{\pm 0.8}$	$76.6_{\pm 1.2}$	$93.9_{\pm 3.1}$	$51.2_{\pm 5.9}$	$85.4_{\pm 1.5}$	$97.1_{\pm 1.0}$	$31.4_{\pm 0.7}$	$78.8_{\pm 5.6}$	$67.3_{\pm 9.2}$	$81.8_{\pm 5.4}$	$74.6_{\pm 3.5}$	$74.0_{\pm 0.7}$
MatchGPT [Beluga2]	70,000	$79.9_{\pm 1.0}$	$78.6_{\pm 1.7}$	$91.4_{\pm 4.4}$	$79.1_{\pm 2.6}$	$86.5_{\pm 3.8}$	$96.0_{\pm 3.1}$	$47.6_{\pm 3.4}$	$83.5_{\pm 6.7}$	$55.6_{\pm 8.0}$	$90.8_{\pm 2.2}$	$77.1_{\pm 2.8}$	$78.7_{\pm 0.7}$
MatchGPT [GPT-4o-Mini]	8,000	$87.2_{\pm 0.6}$	$\underline{88.4}_{\pm 0.4}$	$94.3_{\pm 1.4}$	$87.4_{\pm 1.8}$	$90.8_{\pm 2.8}$	$98.1_{\pm 1.8}$	$60.7_{\pm 1.0}$	$67.5_{\pm 8.7}$	$69.6_{\pm 9.8}$	$95.7_{\pm 1.5}$	$\underline{82.9}_{\pm 1.2}$	$83.9_{\pm 1.4}$
MatchGPT [GPT-3.5-Turbo]	175,000	$75.8_{\pm 3.2}$	$81.9_{\pm 1.9}$	$82.8_{\pm 6.4}$	$62.0_{\pm 10.5}$	$76.0_{\pm 5.7}$	$86.6_{\pm 3.5}$	$39.8_{\pm 2.9}$	$46.6_{\pm 9.4}$	$38.2_{\pm 6.6}$	$70.7_{\pm 6.2}$	$66.0_{\pm 5.7}$	$66.0_{\pm 3.4}$
MatchGPT [GPT-4]	1,760,000	$\mathbf{92.4}_{\pm 0.5}$	$\mathbf{89.1}_{\pm 0.4}$	$96.0_{\pm 1.0}$	$87.9_{\pm 1.1}$	$95.1_{\pm 4.1}$	$97.9_{\pm 4.1}$	$\mathbf{75.0}_{\pm 0.9}$	$82.5_{\pm 2.1}$	$62.9_{\pm 7.8}$	$\mathbf{97.2}_{\pm 3.4}$	$\mathbf{85.1}_{\pm 1.3}$	$\underline{87.4}_{\pm 0.9}$

"domain knowledge [...] highlighting important pieces of the input", since such knowledge is not available in a cross-dataset setting. For Unicorn, we leverage the provided source code and use the instruction version with a DeBERTa [21] base model that can support cross-dataset entity matching. Since their implementation includes only five EM datasets, we modify the code to incorporate all 11 EM datasets for the "leave-one-dataset-out" configuration. All other hyperparameters are kept identical to those in the original implementation.

AnyMatch already complies with the 'leave-one-dataset-out' configuration for its two base models GPT-2 [42] and T5 [43], so we only need to re-run the experiments with different random seeds and serialization strategies. To fully explore the performance boundaries of fine-tuning SLMs, we introduce an additional variant — based on a LLaMA3.2 model[1] with 1.3 billion parameters, which slightly exceeds the threshold we set for categorising PLMs. We adjust the learning rate to $1e^{-6}$ while keeping all other hyperparameters unchanged. Additionally, we do not apply the AutoML boosting and data augmentation with weakly-labelled attribute pairs for this variant, but retain the label balancing operation to ensure that both matching and non-matching pairs are adequately represented in the training data.

Instruction-Tuned or Continual-Tuned Language Models. For Jellyfish, we leverage the publicly available pretrained 13 billion parameter model and prompt format provided by the authors. Unfortunately, the authors used six out of the eleven benchmark datasets during the multi-task training of their model. As a consequence, we cannot fairly evaluate the model in a cross-dataset setting on this data, since it has already seen training data for these tasks. We still report the resulting numbers for completeness, but put them in brackets to indicate that they do not originate from a cross-dataset setup. The original work on TableGPT [29] reports numbers for seven of our included datasets in their zero-shot setting, where the task (entity matching) has been seen during training, but no labelled pairs from the target data were observed. However, since TableGPT is proprietary and not accessible to the academic community, we are unable to evaluate their model under our experimental configuration.

Prompted large language models. Similarly, for the study by Narayan et al. [37], we are unable to conduct the experiments as the GPT-3 model has been deprecated. Therefore, we provide their numbers only as supplementary material in our code repository for reference. For MatchGPT, we leverage prompting without demonstration examples based on the general-complex-force prompt format from [41], which showed the best performance without domain-specific information in the cited study. We evaluate the three variants Mixtral, SOLAR and Beluga2 which are based a set of large open-weight models ranging from 56 billion to 70 billion parameters. Furthermore, we include the latest variants GPT-3.5-Turbo, GPT-4, and GPT-4o-Mini, to ensure up-to-date comparisons. These models are accessed via the commercial API from OpenAI, with parameter sizes assumed to be 175 billion for GPT-3.5-Turbo, 1.76 trillion (8x220B) [53] for GPT-4 and 8 billion for GPT-4o-Mini.

Results and discussion. We list the resulting mean F1 scores and their standard deviation for the 11 datasets in Table 3. The best score per dataset is indicated in bold, while the second-best score is underlined. The six scores for Jellyfish is in brackets to indicate that it was pretrained on this data, which violates the cross-dataset setting. We additionally plot the prediction quality versus the model size (in terms of the number of parameters) in Figure 4. Overall, fine-tuned SLMs have higher variance than LLMs, which we attribute to the larger parameter size providing greater robustness for such prediction tasks. Notably, AnyMatch [LLaMA3.2] achieves the best results, showing predictive quality on par with MatchGPT [GPT-4] — with an average F1 score of 87.5 compared to 87.4 — while using three orders of magnitude fewer parameters. We also observe a strong performance from MatchGPT [GPT-4o-mini], whose score is less than 1% lower than the score of Jellyfish, despite the latter being instruction-tuned with nearly double the parameter size. Additionally, other fine-tuned SLMs, such as AnyMatch [GPT-2], AnyMatch [T5], and Unicorn, perform well, scoring around 80%, surpassing the performance of prompting all three open-source LLMs.

In the following, we present six key findings from this experiment.

Finding 1 – Parameter-free matchers are only competitive with fine-tuned small language models and prompted open language models on specific datasets. As expected, the StringSim baseline yields the lowest F1 score of just 43.5. Notably, while still lagging behind the top-performing methods, the parameter-free approach ZeroER proves competitive on the DBAC and FOZA datasets, which are relatively well-structured. However, it underperforms on several datasets with more free-text content, such as ABT, WDC,

[1] https://huggingface.co/meta-llama/Llama-3.2-1B

AMGO, ITAM, and WAAM. We attribute this to the lengthy and unconventional product descriptions in these datasets, which introduce substantial complexity and are not effectively captured by the similarity differences between matched and mismatched record groups. Nevertheless, the strong performance on specific datasets still suggests potential for developing hybrid methods that combine efficient, parameter-free matchers with other techniques.

Finding 2 – Model-agnostic matchers generally perform better than model-aware matchers. When considering small fine-tuned models, it becomes evident that a model-agnostic approach is more suitable for cross-dataset entity matching (EM) than a model-aware approach. The over 8-point higher performance of Unicorn compared to Ditto demonstrates the importance of designing a prediction head based on representations extracted from PLMs to achieve high-quality predictions. However, despite Unicorn having a relatively complex model trained on extensive data from EM tasks and other tasks (with over 1 million samples), it still falls short when compared to the smaller AnyMatch [GPT-2] and performs only 2.4% better than AnyMatch [T5]. The comparison becomes more complicated when comparing model-aware solutions against model-agnostic and prompt-based methods. While Unicorn outperforms MatchGPT [Mixtral-8x7B], MatchGPT [SOLAR], MatchGPT [Beluga2], and MatchGPT [GPT-3.5-Turbo], it still underperforms compared to MatchGPT [GPT-4o-mini] and MatchGPT [GPT-4]. This suggests that a performance gap remains when prompting advanced large models with a fine-tuned, model-aware matcher, even one designed specifically to support cross-dataset EM.

Finding 3 – Fine-tuned small language models can achieve on-par or even better performance than prompted large language models. Fine-tuning SLMs results in reasonable performance, which in all cases reaches at least the quality level of prompting open models like Mixtral. Even with the smallest fine-tuned model, Ditto, we observe on-par performance to Mixtral and SOLAR. Approaches with a complex model design, such as Unicorn, outperform all open-source LLMs and MatchGPT [GPT-3.5-Turbo] with a large margin, despite the fact that these models have at least two orders of magnitude more parameters. When a model-agnostic matcher is fine-tuned, we observe even stronger results. AnyMatch [LLaMA3.2] with 1.3 billion parameters results in the best performance of 87.5, and slightly surpasses the trillion-parameter model MatchGPT [GPT-4], which gives a score of 87.4. However, MatchGPT [GPT-4] achieves 5 out of the 11 best results, while AnyMatch [LLaMA3.2] secures only two best results and five second-best results. The high performance of fine-tuned SLMs (Figure 4) also gives rise to a potentially very cost-effective deployment of high-quality cross-dataset matchers, since the model size typically dictates the GPU requirements, which are the the main cost factor. We explore this relationship further in Section 4.2

Finding 4 – The commercial GPT series models are well-suited to handling input expressed in domain-specific language. MatchGPT [GPT-4o-Mini] demonstrates decent performance across all datasets, despite having only 8 billion parameters and not being fine-tuned. Its direct competitors AnyMatch [GPT-2] and Unicorn both show similar performance with an around 81 F1 score. An observation on these methods is that MatchGPT [GPT-4o-Mini] consistently and greatly outperforms them on the WDC and WAAM datasets by more than 10 points. This trend also holds in the comparison between MatchGPT [GPT-4] and AnyMatch [LLaMA3.2]. We assume this performance difference caused by the fact that such datasets uses very domain specific language, which is often not grammatically consistent. For example, AnyMatch [GPT-2] misclassifies the product pairs {title: "sumdex slr camera sling pack", category: "mp3 accessories", brand: "sumdex"} and {title: "slr camera sling pack", category: "cases bags", brand: "sumdex"} from AMGO dataset, while MatchGPT [GPT-4o-Mini] correctly matches them.

We attribute this to the LLM's exposure to a more complex and enriched pretraining corpus, which makes it likely to have encountered such textual patterns. In contrast, the smaller model may lack exposure to these sequences and must rely on further fine-tuning, which is constrained by the perplexity of provided training datasets. Another unknown factor in these results is whether the commercial models have encountered the public benchmark datasets during their pretraining. However, this remains impossible to determine, as the training data for these models is not disclosed.

Finding 5 – Overlapping domain datasets have negligible impact in cross-dataset configurations. As shown in Table 3, six datasets share the same domain with at least one other dataset. In our leave-one-dataset-out fine-tuning configurations, when one of these datasets is left out, the training set still includes data samples from the same domain. Intuitively, one might hypothesize that the presence of same-domain data in the training set positively influences predictions for the target dataset. We test for this hypothesis that overlapping domain datasets lead to higher performance, by conducting a two sample t-test between the F1 scores for datasets that share a domain and those that do not. Note that we normalize the F1 scores by subtracting the mean F1 score of a LLM (we use MatchGPT [GPT-3.5-Turbo]), to put all the scores on the same scale. The t-test results in a rejection of the hypothesis, indicating that overlapping domains do not significantly improve performance.

The capability of a LM is determined by two factors: its understanding of general linguistic context and its understanding of downstream tasks. Since the fine-tuning step primarily contributes to the model's task-specific understanding, we can infer that the fine-tuned models involved in experiments have acquired a general understanding of the EM task from the prepared high-perplexity datasets. This understanding can subsequently be transferred across datasets. Therefore, as long as the textual patterns are not significantly different, even if an upcoming dataset is from an unseen domain, the model can still provide accurate responses.

Finding 6 – Both fine-tuned small language models and prompted large language models exhibit insensitivity to skewed datasets. The evaluation in this work includes datasets with varying label distributions, as can be seen from Table 1. We compute the Spearman rank correlation between the predictive quality and label imbalance rate. For all LM baselines, the correlation is approximately 0.15, and never exceeds 0.3, indicating a weak monotonic relationship between the two variables. This result demonstrates that all LMs perform similarly across both major and minor groups, highlighting their insensitivity to skew. From a broader perspective, the correlation for SLMs is slightly smaller than that for LLMs (0.15 vs. 0.18). This can be attributed to the fine-tuning of SLMs with diverse datasets, which enhances their robustness.

Table 4: Average F1 scores for cross-dataset entity matching using different demonstration strategies (best scores in bold per model specification). The model performance is often negatively impacted by demonstrations, except for GPT-4.

		#params	Unseen Target Dataset											
Model	**Demonstrations**	(millions)	ABT	WDC	DBAC	DBGO	FOZA	ZOYE	AMGO	BEER	ITAM	ROIM	WAAM	Mean
GPT-4o-mini	none	8,000	**87.2**$_{\pm 0.6}$	**88.4**$_{\pm 0.4}$	**94.3**$_{\pm 1.4}$	87.4$_{\pm 1.8}$	**90.8**$_{\pm 2.8}$	**98.1**$_{\pm 1.8}$	60.7$_{\pm 1.0}$	**67.5**$_{\pm 8.7}$	**69.6**$_{\pm 9.8}$	**95.7**$_{\pm 1.5}$	**82.9**$_{\pm 1.2}$	**83.9**$_{\pm 1.4}$
GPT-4o-mini	hand-picked	8,000	83.6$_{\pm 2.4}$	86.7$_{\pm 0.8}$	93.9$_{\pm 3.6}$	84.7$_{\pm 2.3}$	89.8$_{\pm 3.6}$	95.6$_{\pm 2.5}$	66.3$_{\pm 1.9}$	60.9$_{\pm 7.9}$	69.3$_{\pm 8.8}$	94.9$_{\pm 2.2}$	82.6$_{\pm 1.3}$	82.6$_{\pm 0.5}$
GPT-4o-mini	random-selected	8,000	86.6$_{\pm 1.9}$	88.0$_{\pm 0.9}$	93.7$_{\pm 2.8}$	**87.7**$_{\pm 1.5}$	90.4$_{\pm 1.7}$	96.6$_{\pm 1.3}$	**66.6**$_{\pm 1.8}$	67.1$_{\pm 3.0}$	68.3$_{\pm 5.6}$	95.4$_{\pm 1.3}$	81.7$_{\pm 1.6}$	83.8$_{\pm 0.6}$
GPT-3.5-Turbo	none	175,000	**75.8**$_{\pm 3.2}$	**81.9**$_{\pm 1.9}$	**82.8**$_{\pm 6.4}$	62.0$_{\pm 10.5}$	**76.0**$_{\pm 5.7}$	**86.6**$_{\pm 3.5}$	39.8$_{\pm 2.9}$	46.6$_{\pm 9.4}$	38.2$_{\pm 6.6}$	**70.7**$_{\pm 6.2}$	**66.0**$_{\pm 5.7}$	66.0$_{\pm 3.4}$
GPT-3.5-Turbo	hand-picked	175,000	59.6$_{\pm 10.1}$	73.9$_{\pm 1.8}$	79.3$_{\pm 1.9}$	55.9$_{\pm 7.4}$	69.5$_{\pm 10.7}$	74.0$_{\pm 4.9}$	38.9$_{\pm 9.0}$	44.5$_{\pm 3.4}$	34.2$_{\pm 7.2}$	57.1$_{\pm 3.3}$	60.2$_{\pm 10.6}$	58.8$_{\pm 2.8}$
GPT-3.5-Turbo	random-selected	175,000	75.7$_{\pm 3.7}$	78.9$_{\pm 3.0}$	82.3$_{\pm 2.6}$	**65.5**$_{\pm 6.8}$	69.8$_{\pm 10.9}$	84.2$_{\pm 5.9}$	**52.1**$_{\pm 5.5}$	**55.9**$_{\pm 2.9}$	**38.4**$_{\pm 5.9}$	69.9$_{\pm 4.5}$	65.1$_{\pm 3.6}$	**67.1**$_{\pm 1.4}$
GPT-4	none	1,760,000	**92.4**$_{\pm 0.5}$	**89.1**$_{\pm 0.4}$	96.0$_{\pm 1.0}$	87.9$_{\pm 1.1}$	95.1$_{\pm 4.1}$	**97.9**$_{\pm 4.1}$	75.0$_{\pm 0.9}$	82.5$_{\pm 2.1}$	62.9$_{\pm 7.8}$	97.2$_{\pm 3.4}$	85.1$_{\pm 1.3}$	87.4$_{\pm 0.9}$
GPT-4	hand-picked	1,760,000	91.3$_{\pm 1.2}$	87.3$_{\pm 1.0}$	**96.9**$_{\pm 1.5}$	**89.2**$_{\pm 1.3}$	**95.7**$_{\pm 3.0}$	97.7$_{\pm 2.8}$	75.1$_{\pm 1.9}$	80.6$_{\pm 4.6}$	72.3$_{\pm 6.2}$	**99.5**$_{\pm 1.0}$	**85.6**$_{\pm 0.6}$	88.3$_{\pm 0.4}$
GPT-4	random-selected	1,760,000	90.4$_{\pm 0.6}$	87.9$_{\pm 0.9}$	96.3$_{\pm 0.5}$	88.6$_{\pm 0.8}$	**95.7**$_{\pm 3.0}$	97.3$_{\pm 2.6}$	**75.3**$_{\pm 1.0}$	**85.1**$_{\pm 1.8}$	**73.2**$_{\pm 3.0}$	99.2$_{\pm 1.0}$	83.2$_{\pm 1.7}$	**88.4**$_{\pm 0.5}$

4.1.1 Benefit of additional demonstrations for prompted LLMs. In our cross-dataset EM setup, prompted LLMs can still be potentially impacted with demonstrations from the transfer data. This consideration is missing in the study on MatchGPT [41]. In order to make sure that our results are not impacted by this gap, we conduct an additional experiment to explore the benefits of providing demonstrations to prompted LLMs.

Experimental setup. We compare three different variants for including demonstrations in MatchGPT. The first variant mirrors the setting used in the previous experiment and does not include any demonstration examples. The second variant utilises three (two non-matching and one matching) manually selected examples from the available transfer learning datasets in its prompt. The third variant contains three randomly selected examples from the available transfer learning datasets to generate prompts. We evaluate MatchGPT with three LLMs: GPT-4o-mini, GPT-3.5-Turbo, and GPT-4.

Results and discussion. Firstly, for the GPT-4o-Mini and GPT-3.5-Turbo models, prompting with demonstrations in cross-dataset EM scenarios tends to degrade model performance in most cases, regardless of the demonstration strategy. Specifically, prompting without demonstrations achieves the highest scores in 10 out of 11 datasets for GPT-4o-Mini and in 7 out of 11 datasets for GPT-3.5-Turbo. At a first glance, these findings may seem counterintuitive and appear to contradict the results of [37], which demonstrated that manually selected demonstrations by experts can improve model performance of GPT-3, while randomly selected demonstrations introduce greater sensitivity to the model. Actually, these findings effectively complement the original conclusion. In Narayan's study, the demonstrations for each test sample were drawn from the same dataset, whereas our cross-dataset setting required the selected demonstrations to come from out-of-distribution datasets. Consequently, for models like GPT-3.5 and GPT-4o-Mini, providing demonstrations can confuse the model due to the out-of-distribution cases in the context, leading to worse performance. As for the more advanced model GPT-4, demonstrations can lead to subtle performance improvements. This suggests that this model possesses more general knowledge of the EM task and can effectively learn from context examples even when they come from different datasets. Lastly, randomly selected demonstrations are generally more helpful to the model compared to manually selected ones. We attribute this to the fact that manually selected demonstrations are few in number and closely tied to specific datasets, making them less effective at form a helpful context for the current test sample in cross-dataset scenarios. Consequently, the random selection strategy tends to outperform the manual selection strategy.

4.2 Inference Throughput & Deployment Cost

The goal of the following two experiments is to assess the trade-off between the deployment cost and prediction quality of the cross-dataset EM approaches under consideration. The ability to deploy cross-dataset EM models in a cost-efficient and scalable way is especially important for the use cases discussed in Section 2.1, such as data integration services in the cloud or deduplication as a data cleaning step in machine learning pipelines [30, 57].

We focus on throughput as a measure of computational performance, as the input for entity matching are typically large candidate sets of potentially matching pairs, which have to be processed in batch. Since we focus on EM with language models, we adopt the common metric of tokens per second [3]. Unfortunately, we cannot properly evaluate the throughput of commercial models like GPT-4 since they are only accessible via proprietary APIs. It is unknown with which hardware they are run, and they are most likely also deployed for multi-tenancy, which makes performance evaluation even more difficult.

Therefore, we adopt the following methodology for our performance experiments: We measure and compare the throughput (in terms of tokens per second on a given hardware setup) of the matchers which leverage publicly available open-weight models (Section 4.2.1). Subsequently, we compare inference with the open-weight models to inference with proprietary model APIs in terms of cost rather than throughput. A common metric is the dollar price per 1,000 tokens, which is available for commercial models from OpenAI. We estimate this cost for AnyMatch, Ditto, Unicorn, Jellyfish, MatchGPT and their respective open-weight models based on the observed throughput numbers and the hourly cost of an appropriate cloud instance in the Amazon Web Services Cloud (Section 4.2.2).

4.2.1 Inference Throughput. The goal of our first experiment is to measure the inference throughput in terms of tokens per second of AnyMatch with three variants, Ditto, Unicorn, the LLM-based methods Jellyfish (which uses the LLaMA2-13B model [49]) and MatchGPT with three open-weight models (Mixtral-8x7B, SOLAR, Beluga2).

Experimental setup. We deploy each matcher (in combination with a given model) with exclusive access to a machine with four A100 GPUs with 40 GB GPU RAM in a large academic HPC cluster. Note that the A100 GPU is a common choice for ML, is the most powerful hardware available to us in academic context, and also constitutes a common choice in cloud instances designed for ML workloads. We leverage implementations based on PyTorch and the transformers library. We deploy quantised (16-bit precision) versions of the models and use model parallelism to distribute a model over multiple GPUs if it cannot fit into the 40 GB memory of a single A100 GPU. We leverage the DBGO dataset here, since it is the largest dataset from our evaluation and proceed as follows. We first determine the maximum batch size usable per model by testing exponentially growing batch sizes and checking for memory issues. Next, we measure the inference time for 100 batches (based on the determined maximum batch size) via the `torch.utils.benchmark` package from PyTorch and compute the throughput in tokens/s based on the observed inference times. If a method does not use all four GPUs, we extrapolate its throughput to the full machine based on the number of GPUs used, as our inference is embarrassingly parallel.

Results and discussion. We list the required memory per model, the corresponding maximum usable batch size and the achieved throughput in Table 5. The models used by Ditto, AnyMatch, Unicorn, and Jellyfish can fit into the 40 GB memory of a single A100 GPU. Mixtral-8x7B requires model parallelism with two such GPUs, while SOLAR and Beluga2 need to be distributed over all four A100 GPUs. The differences in size and the required model parallelism result in vast differences in the maximum achievable batch size and throughput.

Impressively, Ditto has the highest throughput across all included method, of 862,001 tokens per second, which is is 1,146 and 798 times larger than that of the open-sourced large language models SOLAR and Beluga2, respectively. When excluding the Jellyfish method, we can notice that the throughput of all fine-tuned small language models exceeds two orders of magnitude compared to that of the larger language models. This can be attributed to several factors. Firstly, the memory requirements are significant, as larger models like Mixtral-8x7B, Beluga2, and SOLAR cannot fit into a single A100 GPU, necessitating the use of model parallelism across multiple GPUs. This setup can hinder throughput because the model activations must be copied to the memory of the other GPUs. Secondly, the maximum batch size that can be accommodated is another constraint on throughput. Typically, a smaller parameter size allows for a higher batch size within the same memory configuration.

Models like Ditto, AnyMatch [GPT-2], and AnyMatch [T5] can accommodate 8,192 examples in a single batch, which is two orders of magnitude larger than that of larger language models like Mixtral-8x7B, Beluga, and SOLAR, which can only handle between 32 and 64 examples. Interestingly, the structural design of the model also influences its maximum accommodatable batch size and, consequently, the throughput. For instance, Unicorn employs a mixture of expert design, which has a similar number of parameters compared to Ditto and AnyMatch [GPT-2]. However, its batch size is only half that of the former two methods, resulting in a throughput that is only one-fourth to one-third of theirs. A similar situation is observed between Mixtral-8x7B and Beluga2.

Table 5: Throughput in tokens/s with 4xA100 (40GB) GPUs fo open-weight LLMs employed by various EM approaches.

Model	Used by	#params (millions)	RAM (GB)	batch size	Throughput (tokens/s)
BERT	Ditto	110	0.21	8,192	**862,001**
GPT-2	AnyMatch	124	0.26	8,192	693,999
DeBERTa	Unicorn	143	0.27	4,096	216,396
T5	AnyMatch	220	0.54	8,192	530,656
LLaMA3.2	AnyMatch	1,300	2.30	4,096	264,952
LLaMA2-13B	Jellyfish	13,000	24.46	128	26,721
Mixtral-8x7B	MatchGPT	56,000	73.73	32	2,108
Beluga2	MatchGPT	70,000	128.64	32	1,079
SOLAR	MatchGPT	70,000	128.64	64	752

4.2.2 Inference Cost. The goal of the last discussion is to compare fine-tuned small language models against entity matching with approaches like MatchGPT that use commercial models from OpenAI. As discussed, we cannot reliably measure the throughput for these models since they are deployed behind proprietary APIs on unknown hardware. As a consequence, we compare the dollar cost of inference with these models to the dollar cost of inference with Ditto, AnyMatch, Unicorn, and MatchGPT with the open-weight models Mixtral-8x7B, SOLAR and Beluga2.

Table 6: Cost per 1K tokens for EM with proprietary models, compared to a deployment scenario with open-weight models on a p4d.24xlarge instance in the AWS cloud or via the together.ai platform.

Method & model	Cost for 1K tokens	Deployment scenario
MatchGPT [GPT-4]	$0.015	OpenAI Batch API
MatchGPT [SOLAR]	$0.0009	Hosting on Together.ai
MatchGPT [Beluga2]	$0.0009	Hosting on Together.ai
MatchGPT [GPT-3.5-Turbo]	$0.00075	OpenAI Batch API
MatchGPT [Mixtral-8x7B]	$0.00063	4x on p4d.24xlarge
MatchGPT [GPT-4o-Mini]	$0.000075	OpenAI Batch API
Jellyfish	$0.000025	8x on p4d.24xlarge
Unicorn[DeBERTa]	$0.000012	8x on p4d.24xlarge
AnyMatch[LLaMA3.2]	$0.000010	8x on p4d.24xlarge
AnyMatch[T5]	$0.0000050	8x on p4d.24xlarge
AnyMatch[GPT-2]	$0.0000038	8x on p4d.24xlarge
Ditto[Bert]	$0.0000031	8x on p4d.24xlarge

Note that we cannot include GPT-3 and TableGPT in this discussion since we could not estimate its cost due to the deprecation of the used models. We also do not include Jellyfish in this discussion, since we cannot reliably compute its average F1 score, as it has seen several of the evaluation datasets at training time, which violates our cross-dataset setting (as discussed in Section 4.1).

Setup. We lookup the costs for the commercial models from OpenAI at https://openai.com/api/pricing/. As of December 2024, batch inference with the GPT-4 model costs $0.015 per 1,000 tokens and inference with GPT-3.5-Turbo-0125 costs $0.00075 per 1,000 tokens. Note that these models have different costs for input and output tokens; we use the cheaper input token cost, since entity matching is modelled as sequence classification task,

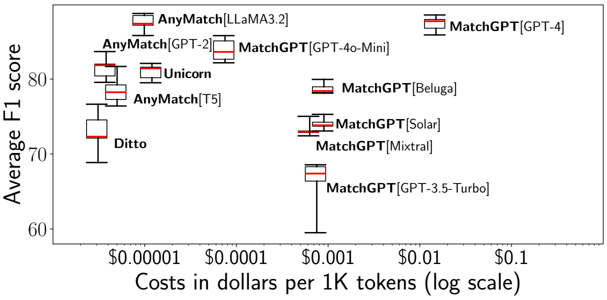

Figure 3: Deployment cost versus prediction quality.

which only generates output with a single word. We estimate the cost for fine-tuned small language models and the open-weight models as follows. We assume that such a model is deployed on a cloud instance that is constantly used for inference (e.g., as part of the use cases described in Section 2.1). We use the cost for a `p4d.24xlarge` instance[2] from the Amazon Web Services cloud as a reference. This machine is designed for ML workloads and comes with eight A100 (40GB) GPUs (exactly twice the amount of GPUs which we used for our throughput experiment). As of December 2024, such a machine has an hourly cost of $19.22 in a scenario where the instance is reserved for a year (which would be common in a corporate setup). Since the cloud instance has the exact same type of GPU (only twice the amount), we can extrapolate our throughput numbers from Section 4.2.1 to this machine by simply doubling them, as inference in entity matching is an embarrassingly parallel workload. We therefore estimate the cost per 1,000 tokens for models deployed on this machine as $(p/(2 \cdot t_m \cdot 3600)) \cdot 1000$ where p is the hourly instance price, t_m is the throughput in tokens/s observed for model m and 2 is the extrapolation factor from our previous experiments (as the cloud instance has twice the amount of GPUs). For the open-weight models, we additionally lookup the hosting price on the cloud platform `together.ai`[3] and choose this option if the resulting price per 1,000 tokens would be lower than our self-hosting setup (e.g., because a more favourable GPU can be chosen).

Results and discussion. We list the resulting costs per method and model in descending order in Table 6 . For each entry, we also mention the chosen cheapest deployment scenario, e.g. whether we assume that the OpenAI API is used, whether we assume that the model is hosted on together.ai, or whether we assume that the model is deployed x-times on a `p4d.24xlarge` instance in AWS.

As observed, the lowest cost is provided by `Ditto [Bert]`, which is 4,838 times cheaper than the most expensive solution, `MatchGPT [GPT-4]`. `AnyMatch [GPT-2]` ranks second in terms of cost-efficiency. The `Unicorn` method and `AnyMatch [LLaMA3.2]` have similar costs, both being three times more expensive than `Ditto`. While all fine-tuned small language models exhibit low inference costs, there are also cost-effective solutions within the large language model (LLM) category. Notably, `Jellyfish` and `MatchGPT [GPT-4o-Mini]` maintain a cost comparable to `Unicorn`, being an order of magnitude cheaper than most other LLMs except `MatchGPT [GPT-4]`. In fact, their costs are two orders of magnitude lower compared to `MatchGPT [GPT-4]`.

Trade-off between deployment cost and prediction quality. Given the distinct cost and quality variance among different matchers, we seek to find a trade-off between these two dimensions. This balance is crucial for designing cost-efficient and scalable EM approaches, e.g. for a data integration service in the cloud. For that. we plot the average F1 score achieved versus the estimated cost for 1,000 tokens from the analysis in Figure 3.

The method with the highest performance and lowest cost should ideally appear in the top-left corner of Figure 1. However, as observed, no single method excels in both dimensions. Among the evaluated approaches, `AnyMatch [LLaMA3.2]` strikes the best balance. For systems with a budget of less than $0.00005 per 1K tokens, all `AnyMatch` models and `Unicorn` are viable options. When the budget increases to $0.000075 per 1K tokens, the selection can expand to include `MatchGPT [GPT-4o-Mini]`. This choice offers more stable and higher performance, especially for inputs with highly specific or unconventional language expressions, as discussed in Finding 4.

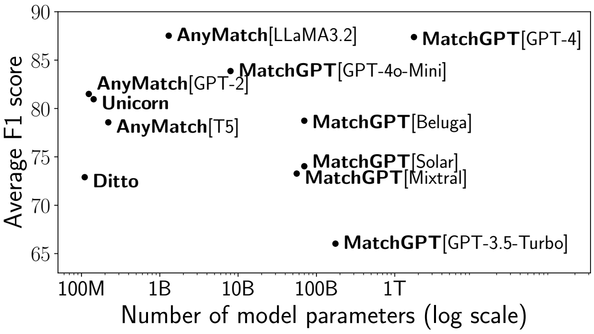

Figure 4: Model size versus prediction quality. Fine-tuned small models perform on par with prompted LLMs.

5 LESSONS & DISCUSSION

We discuss several insights and limitations from our study.

Data-centric approaches outperform model-centric ones. A significant finding from this study is that data preparation tends to have a greater impact on predictive quality than model design, as shown in Section 4.1. This insight is derived by the results from models like `AnyMatch`, and fine-tuned approaches such as `Ditto` and `Unicorn`. Notably, when the model parameter size is comparable, `AnyMatch` achieves an on par performance score of 81.5 compared to the more intricately designed model `Unicorn`, which scores 81.0. When these data-centric strategies are applied to larger models for fine-tuning, the performance can improve even further, as demonstrated by the best solution achieving an F1 score of 87.5. This insight aligns with the paradigm shift from optimising model structure to focusing on preparing more informative training corpora for language models [25, 42].

Fine-tuned small models perform on par with prompted large models. Another key takeaway stem from the comparison between fine-tuning smaller LMs and prompting large models, like the GPT series, for cross-dataset EM. The simplest approach, `Ditto`, achieves an F1 score of 72.9, on par with most prompted large models, except `GPT-4` and `GPT-4o-Mini`, which score 87.4 and 83.9, respectively. Fine-tuning a 124-million-parameter `GPT-2` model yields an F1 score of 81.5, not far away from `GPT-4o-Mini`, while fine-tuning the 1.3-billion-parameter `LLaMA3.2` model can outperform or achieve comparable performance to all prompted large models. Moreover, fine-tuning smaller models offers distinct cost advantages at least two orders of magnitude for deployment. This becomes particularly important when considering the prohibitive costs associated with large numbers of API calls to models like `GPT-4` in industry settings. Fine-tuning smaller models thus presents a more scalable and cost-efficient solution for tasks like cross-dataset EM, without compromising on performance. A future direction for this research could involve testing whether fine-tuning SLMs continues to outperform prompted LLMs in other table-relevant tasks.

Lack of Impact of Demonstrations. When comparing prompting techniques, we find that prompting without demonstrations yields better performance in most cross-dataset scenarios with

[2] https://aws.amazon.com/ec2/instance-types/p4/

[3] Available at https://www.together.ai/pricing, accessed in December 2024

GPT-4o-Mini and GPT-3.5-Turbo. For example, prompting GPT-4o-Mini without demonstrations achieves the best scores in 9 out of 11 datasets across different prompting methods. For the more advanced model GPT-4, the impact of demonstrations is subtle.

Interestingly, prompting with randomly demonstrations outperform manually chosen ones. This finding appears to contradict the claim made by [37], which suggests that prompting with hand-picked examples from the same dataset as the target data improves model performance, while randomly selected examples lead to increased sensitivity and decreased quality. In fact, our experimental results can be viewed as complementary to their conclusion in a cross-dataset setting. In our case, demonstration examples were sourced from datasets different from the target data, making the situation more analogous to the random case described. Such out-of-distribution demonstrations may confuse the model, leading to less accurate predictions, particularly for less capable models. Additionally, as manually selected examples are closely tied to specific datasets, they are less effective at forming a helpful context for predicting target samples, which explains why they generally perform worse than random selections.

Recommendation for practitioners. For practitioners aiming to develop a scalable cloud service for EM, we recommend using AnyMatch [LLaMA3.2] when sufficient transfer data is available. This configuration strikes an excellent balance between performance and cost-efficiency. Furthermore, it allows the control over the underlying model—a critical advantage, considering the frequent deprecation of commercial models. In scenarios where transfer data is limited or unavailable, MatchGPT [GPT-4o-Mini] serves as a strong alternative, delivering robust performance across various datasets. Although it may incur a higher cost (seven times greater at $0.075 compared to $0.01 per million tokens), its capability to effectively handle domain-specific language makes it a valuable option for applications requiring enhanced accuracy and stability.

5.1 Limitations and Future Work

We discuss a set of limitations of our study and outline directions for future work.

Potential data leakage issues. As we aim to evaluate the cross-dataset capability of various LLM-based methods, it is crucial to discuss potential data leakage issues that may arise during the pretraining phase of LMs. Given that most LMs are trained on a vast corpus of web data, there is a risk that this data may overlap with the datasets on which we are evaluating. Unfortunately, the training data used for the GPT series models, including the open-weight model GPT-2, is not publicly available. As a result, the academic community cannot verify whether there are any data leakage issues when working with these models.

The training corpora of some open-weight models training corpus are available, allowing us to validate if a dataset was in included in their training corpus. We are only able to run such an analysis for T5 [43]-one of the base models applied by AnyMatch. T5 is pretrained on the C4 dataset, a cleaned version of the Common Crawl web corpus. We download the 'C4/en' version (350 GiB) from HuggingFace[4] and conduct a sanity check. Each data sample in this dataset includes a URL field that specifies the source of the textual data. We thoroughly examined whether the source repositories of our datasets are included in this field. Based on our comparison, we found no evidence that the EM data were used during the pretraining of T5. Moreover, we conducted a separate analysis on dataset pairs by looking at the result size of natural joins between them to ensure there is no overlap. This analysis confirmed that there is zero tuple overlap between every pair of datasets.

Task-specific data required for fine-tuning. Another limitation is that, while fine-tuning smaller models appears cost-effective, it requires access to task-specific data for fine-tuning, which may not always be easy to collect for practical cross-dataset applications. A future direction of research is to investigate if other matching-relevant data such as samples for schema matching, column alignment, etc. (as mentioned in [50]) can help if task-specific data is missing.

Lack of comparison with RAG approaches. Another future direction for cross-dataset EM is Retrieval-Augmented Generation (RAG). RAG methods, which enhance language models by retrieving relevant information from external knowledge bases, could provide additional performance improvements, particularly in scenarios requiring high precision. While our work focuses on fine-tuning and prompting, future studies should investigate whether integrating RAG would improve the effectiveness of prompting with demonstrations in our cross-dataset EM task.

Lack of real-world applicability. As real-world EM datasets are typically costly to obtain, our work is limited to evaluations on openly available benchmark datasets. While the 11 datasets used in this study include noisy data, such as unformatted or missing values, they may not fully capture the complexities of real-world applications. Extending this line of research to investigate how existing EM methods perform in real-world scenarios would be a valuable and intriguing direction for future work.

6 RELATED WORK

Entity matching is a well-studied problem with several surveys and benchmarks. To the best of our knowledge, no existing benchmark covers large language models and their cost in a cross-dataset setting however.

Benchmarks and Studies on Entity Matching. An extensive analysis of EM on real-world datasets has been presented by [26] already more than a decade ago, while more recent studies [9, 36] focuses on the potential of deep learning model for the entity matching problem. However, the cross-dataset setting is not considered. A crucial part of real-world EM systems are blocking and filtering techniques to reduce the number of candidate pairs to match [8, 20, 39]. We focus on the subsequent matching methods, which can be combined with any blocking or filtering technique. [35] study an active learning setting for EM where a small number of labelled examples can be dynamically collected, which in contrast to our setting, requires the active involvement of human labellers. [12] present a systems vision for end-to-end matching solutions that cover the whole matching pipeline and assist users in selecting methods for individual steps.

A recent study [47] on the fairness of entity matchers uncovered problems in light of the coverage of certain demographic groups or with the similarity characteristics of certain names. The REIN [1] and CleanML [30] benchmarks investigate the impact of data errors and cleaning techniques (including entity matching in the form of deduplication) on the downstream performance of ML models. Furthermore, there are several critical assessments of the difficulty of the entity matching task [27, 36, 38]. We interpret

[4] https://huggingface.co/datasets/allenai/c4

their finding that many EM datasets are easy to match as evidence for the potential of cross-dataset matchers and they feasibility of EM systems that require minimal human intervention.

Approaches without Cross-Dataset Capabilities. In addition to the EM solutions detailed in Section 3, we discuss a set of methods for entity matching, which are not able to adapt to the cross-dataset setting on unseen data, since their feature encoding depends on the schema of the target data to match. Magellan [12] focuses on building an end-to-end system for entity matching, based on classical machine learning methods. *GNEM* [7] employs a graph neural network approach to entity matching, where each node represents an entity pair and encodes semantic information and interactions. *HierMatch* [17] introduces a novel approach to entity matching by constructing a hierarchical structure that progresses from the token level to the attribute level, and finally to the entity level. *DeepMatcher* [56] is a transformer-based neural network for entity matching with various components to improve classification performance across three distinct types of input: structured, text, and dirty data. *MCAN* [59] extends the DeepMatcher model with an attention mechanism after the attribute matching phase. The methods proposed in [19, 24] adopt an active learning strategy to manually identify informative samples for model training, that improve matching performance when labelled.

7 CONCLUSION

We presented the first comprehensive and systematic study on cross-dataset entity matching with large and small language models, where we evaluated 14 matchers on 11 benchmark datasets. With respect to prediction quality, we found that fine-tuned small models can perform comparable to prompted large models and that data-centric approaches outperform model-centric approaches. Furthermore, our analysis of the deployment cost indicated that well-performing small models can be deployed at an orders of magnitude lower cost than comparably performing approaches with large commercial models.

Reproducibility. We have released all datasets, code, and models necessary to reproduce the results in this work at https://github.com/Jantory/cross-dataset-em-study. The fine-tuning process for the small language models is based on the original implementation, as detailed in Table 2. For all the small language models we fine-tuned, we provide both the adjusted code from their source implementations and the complete scripts for fine-tuning and evaluation. Furthermore, the dedicated notebook used for designing prompts and performing inference with the large language models is also open-sourced.

Acknowledgements. This research was supported by the University of Amsterdam Data Science Center and the project CaRe-NLP (NGF.1607.22.014) - AiNed Fellowship Beurzen 2022-2023 which is (partly) financed by the Dutch Research Council (NWO).

REFERENCES

[1] Mohamed Abdelaal, Christian Hammacher, and Harald Schoening. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. *Proceedings of the VLDB Endowment (PVLDB)* (2023).

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.

[3] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed inference and fine-tuning of large language models over the internet. *Advances in Neural Information Processing Systems* 36 (2024).

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 1335–1349.

[6] Peter Baile Chen, Fabian Wenz, Yi Zhang, Moe Kayali, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2024. BEAVER: An Enterprise Benchmark for Text-to-SQL. *arXiv preprint arXiv:2409.02038* (2024).

[7] Runjin Chen, Yanyan Shen, and Dongxiang Zhang. 2021. GNEM: a generic one-to-set neural entity matching framework. In *Proceedings of the Web Conference 2021*. 1686–1694.

[8] Peter Christen. 2011. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering* 24, 9 (2011), 1537–1555.

[9] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–42.

[10] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n.d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] AnHai Doan, Pradap Konda, Paul Suganthan GC, Yash Govind, Derek Paulsen, Kaushik Chandrasekhar, Philip Martinkus, and Matthew Christie. 2020. Magellan: toward building ecosystems of entity matching solutions. *Commun. ACM* 63, 8 (2020), 83–91.

[13] Till Döhmen, Radu Geacu, Madelon Hulsebos, and Sebastian Schelter. 2024. SchemaPile: A Large Collection of Relational Database Schemas. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[14] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.* 11, 11 (July 2018), 1454–1467. https://doi.org/10.14778/3236187.3236198

[15] Grace Fan, Roee Shraga, and Renée J Miller. 2024. Gen-T: Table Reclamation in Data Lakes. *arXiv preprint arXiv:2403.14128* (2024).

[16] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large Language Models on Tabular Data–A Survey. *arXiv preprint arXiv:2402.17944* (2024).

[17] Cheng Fu, Xianpei Han, Jiaming He, and Le Sun. 2021. Hierarchical matching network for heterogeneous entity resolution. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 3665–3671.

[18] Yihan Gao, Silu Huang, and Aditya Parameswaran. 2018. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *Proceedings of the 2018 International Conference on Management of Data*. 943–958.

[19] Bar Genossar, Avigdor Gal, and Roee Shraga. 2023. The battleship approach to the low resource entity matching problem. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 1–25.

[20] Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, Donatello Santoro, Enzo Veltri, et al. 2024. Similarity Measures For Incomplete Database Instances. In *Advances in Database Technology-EDBT*. Vol. 27. OpenProceedings. org, 461–473.

[21] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654* (2020).

[22] Zezhou Huang and Eugene Wu. 2024. Relationalizing Tables with Large Language Models: The Promise and Challenges. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 305–309.

[23] Nick Hynes, D. Sculley, and Michael Terry. 2017. The Data Linter: Lightweight Automated Sanity Checking for ML Data Sets. *Machine Learning Systems workshop at NeurIPS* (2017).

[24] Arjit Jain, Sunita Sarawagi, and Prithviraj Sen. 2021. Deep indexed active learning for matching heterogeneous entity representations. *arXiv preprint arXiv:2104.03986* (2021).

[25] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei.

2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[26] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 484–493.

[27] Manuel Leone, Stefano Huber, Akhil Arora, Alberto García-Durán, and Robert West. 2022. A critical re-evaluation of neural methods for entity alignment. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1712–1725.

[28] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4213–4216.

[29] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[30] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A study for evaluating the impact of data cleaning on ml classification tasks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 13–24.

[31] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584* (2020).

[32] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. *arXiv preprint arXiv:2405.14696* (2024).

[33] Kyle Luoma and Arun Kumar. 2024. SNAILS: Schema Naming Assessments for Improved LLM-Based SQL Inference. https://adalabucsd.github.io/papers/TR_2025_SNAILS.pdf.

[34] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1930–1939.

[35] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A comprehensive benchmark framework for active learning methods in entity matching. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 1133–1147.

[36] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 international conference on management of data*. 19–34.

[37] Avanika Narayan et al. 2022. Can Foundation Models Wrangle Your Data? *PVLDB* (2022).

[38] George Papadakis, Nishadi Kirielle, Peter Christen, and Themis Palpanas. 2024. A critical re-evaluation of benchmark datasets for (deep) learning-based matching algorithms. *ICDE* (2024).

[39] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2019. A survey of blocking and filtering techniques for entity resolution. *arXiv preprint arXiv:1905.06167* (2019).

[40] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2024. Qatch: Benchmarking sql-centric tasks with table representation learning models on your data. *Advances in Neural Information Processing Systems* 36 (2024).

[41] Ralph Peeters and Christian Bizer. 2023. Entity matching using large language models. *arXiv preprint arXiv:2310.11244* (2023).

[42] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019), 9.

[43] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[44] Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. 2024. A review on large Language Models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access* (2024).

[45] Amazon Web Services. 2022. AWS Glue. https://aws.amazon.com/glue/.

[46] Amazon Web Services. 2022. Teaching the Find Matches transform. https://docs.aws.amazon.com/glue/latest/dg/machine-learning-teaching.html.

[47] Nima Shahbazi, Nikola Danevski, Fatemeh Nargesian, Abolfazl Asudeh, and Divesh Srivastava. 2023. Through the Fairness Lens: Experimental Analysis and Evaluation of Entity Matching. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3279–3292.

[48] Michael Stonebraker, Ihab F Ilyas, et al. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.* 41, 2 (2018), 3–9.

[49] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[50] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[51] David Vos, Till Döhmen, and Sebastian Schelter. 2022. Towards parameter-efficient automation of data wrangling tasks with prefix-tuning. In *NeurIPS 2022 First Table Representation Workshop*.

[52] Jin Wang, Yuliang Li, Wataru Hirota, and Eser Kandogan. 2022. Machop: an end-to-end generalized entity matching framework. In *Proceedings of the Fifth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. 1–10.

[53] Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024. Mixture-of-Agents Enhances Large Language Model Capabilities. *arXiv preprint arXiv:2406.04692* (2024).

[54] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xuanang Chen, Xianpei Han, Hao Wang, Zhenyu Zeng, and Le Sun. 2024. Match, Compare, or Select? An Investigation of Large Language Models for Entity Matching. *arXiv preprint arXiv:2405.16884* (2024).

[55] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1149–1164.

[56] Tao Xie, Kun Dai, Ke Wang, Ruifeng Li, and Lijun Zhao. 2024. Deepmatcher: a deep transformer-based network for robust and accurate local feature matching. *Expert Systems with Applications* 237 (2024).

[57] Doris Xin, Hui Miao, Aditya Parameswaran, and Neoklis Polyzotis. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. In *Proceedings of the 2021 International Conference on Management of Data*. 2639–2652.

[58] Reynold Xin. 2024. What's new since SIGMOD 1985? - Perspective from a decade of building Databricks. https://www.bifold.berlin/news-events/events/view/event-details/perspective-from-a-decade-of-building-databricks.

[59] Dongxiang Zhang, Yuyang Nie, Sai Wu, Yanyan Shen, and Kian-Lee Tan. 2020. Multi-context attention for entity matching. In *Proceedings of The Web Conference 2020*. 2634–2640.

[60] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *arXiv preprint arXiv:2312.01678* (2023).

[61] Yi Zhang and Zachary G Ives. 2020. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1951–1966.

[62] Zeyu Zhang, Paul Groth, Iacer Calixto, and Sebastian Schelter. 2024. AnyMatch–Efficient Zero-Shot Entity Matching with a Small Language Model. *arXiv preprint arXiv:2409.04073* (2024).