AnyMatch – Efficient Zero-Shot Entity Matching with a Small Language Model

Zeyu Zhang[1,2], Paul Groth[1], Iacer Calixto[2,1], Sebastian Schelter[3]

{z.zhang2, p.t.groth}@uva.nl, i.coimbra@amsterdamumc.nl, schelter@tu-berlin.de
[1]University of Amsterdam [2]Amsterdam UMC [3]BIFOLD & TU Berlin

Abstract

Entity matching (EM)—identifying whether two records refer to the same entity—is critical in data integration. Many EM methods rely heavily on labelled examples, limiting their applicability in real world settings. We address the challenging task of zero-shot entity matching, where no labelled examples are available for an unseen target dataset. Our approach, AnyMatch, leverages a fine-tuned GPT-2 model, enhanced with novel data selection and augmentation techniques within a transfer learning framework. This design enables AnyMatch to achieve predictive performance competitive with much larger language models while providing substantial efficiency gains. Extensive evaluations across nine benchmark datasets and comparisons with thirteen baselines reveal that AnyMatch attains the second-highest overall F1 score, outperforming multiple models with hundreds of billions of parameters. Additionally, AnyMatch offers significant cost advantages: its average prediction quality is within 4.4% of the proprietary trillion-parameter MatchGPT model, yet requires four orders of magnitude fewer parameters and achieves a 3,899-fold reduction in inference cost (in dollars per 1,000 tokens).

1 Introduction

Entity matching (EM), often referred to as entity resolution, is the problem of determining whether two records refer to the same real-world entity. EM is a well-studied problem (Li et al. 2020; Doan et al. 2020; Chen, Shen, and Zhang 2021; Fu et al. 2021; Papadakis et al. 2024; Wang et al. 2022; Mudgal et al. 2018) for its high practical importance in data integration (Abedjan et al. 2016; Stonebraker, Ilyas et al. 2018; Gao, Huang, and Parameswaran 2018; Zhang and Ives 2020; Huang and Wu 2024).

A typical restriction in entity matching scenarios is the dependence on labelled examples. A less restrictive yet more challenging setting is *zero-shot entity matching* (Wu et al. 2020), where the matcher must handle a completely unseen target dataset—without access to labels, schema, or any meta information. Zero-shot matchers are essential to improve data integration services in the cloud (e.g., AWS Glue (Services 2022)), providing automated integration capabilities on enterprise data lakes (Vos, Döhmen, and Schelter 2022). Zero-shot matchers can also be applied for duplicate detection (Hynes, Sculley, and Terry 2017) as part of data cleaning in machine learning pipelines (Abdelaal, Hammacher, and Schoening 2023; Li et al. 2021), and are also valuable as a primitive for entity alignment in tasks such as table reclamation (Fan, Shraga, and Miller 2024).

Presented at the Workshop on Preparing Good Data for Generative AI: Challenges and Approaches (Good-Data) in conjunction with AAAI 2025. The authors retain the copyright.

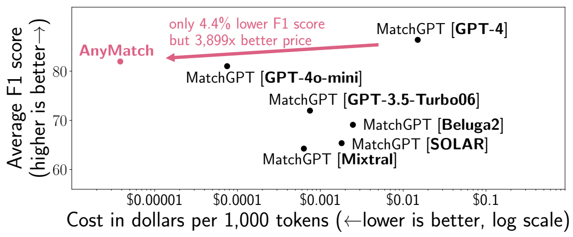


Figure 1: The average F1 score of AnyMatch is only 4.4% lower than the state-of-the-art method `MatchGPT` with `GPT-4`, which incurs a 3,899x higher inference cost in dollars per 1,000 tokens.

Recently, the promising capabilities of large language models (LLMs) (Fernandez et al. 2023) have led to a resurgence of research on EM (Narayan et al. 2022; Peeters and Bizer 2023; Zhang et al. 2023; Li et al. 2024). Approaches such as `MatchGPT` (Peeters and Bizer 2023) or `TableGPT` (Li et al. 2024) show promising results for zero-shot entity matching by prompting LLMs (Narayan et al. 2022). However, these approaches rely on extremely large proprietary models with hundreds of billions or even trillions of parameters (Wang et al. 2024), 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"* (Liu et al. 2024) at a high cost imposed by their pay-per-token model, which results in prices of *"5 USD for processing just 5MB of data"* (Liu et al. 2024). LLM-based entity matchers inherit this high computational cost, which severely limits their scalability and applicability.

To address these challenges, we propose AnyMatch, a cost-efficient approach for zero-shot entity matching on unseen target data. AnyMatch employs a trainable matcher that frames zero-shot entity matching as a transfer learning task. The matching process is handled as a sequence classification problem, as introduced in prior work (Li et al. 2020). In short, AnyMatch fine-tunes the decoder-only language model `GPT-2` (Radford et al. 2019) on carefully curated data samples. As part of our approach, we leverage two novel data selection techniques:

- We employ a boosting technique that leverages an *AutoML filter to identify and prioritise difficult examples*. These challenging instances often represent edge cases where the model tends to perform poorly. Focusing on these difficult examples improves overall robustness and accuracy.
- We *augment the fine-tuning data with attribute-level samples* to accommodate for the structural mismatch between text data and relational data without column order.

We conduct an extensive evaluation of the prediction quality of our approach against *thirteen baselines on nine benchmark datasets*. AnyMatch achieves the second-highest F1 score overall and outperforms twelve baselines (including LLMs with hundreds of billions of parameters), often by a large margin of more than ten percent in F1 score. Moreover, we evaluate the computational performance and deployment cost of AnyMatch in comparison to existing matchers based on large language models, as illustrated in Figure 1. We find that AnyMatch exhibits attractive performance characteristics: its average prediction quality is within 4.4% of the state-of-the-art method `MatchGPT` based on `GPT-4`, yet the latter requires four orders of magnitude more parameters and incurs a 3,899 times higher inference cost (in dollars per 1,000 tokens).

2 Problem Statement

The entity matching (EM) problem is to predict whether the pair of records (r_l, r_r) with $r_l \in \mathrm{R}_{\text{left}}$ and $r_r \in \mathrm{R}_{\text{right}}$ refers to the same real-world entity or not. R_{left} and $\mathrm{R}_{\text{right}}$ denote two input relations with k aligned attributes $A = \{a_1, \ldots, a_k\}$. We refer to these relations as target dataset $\mathcal{D}_{\text{target}}$. EM is often modelled as a binary classification task with a labelled training set $\mathcal{D}_{\text{train}} \subset \mathrm{R}_{\text{left}} \times \mathrm{R}_{\text{right}} \times \{0, 1\}$. State-of-the-art approaches (Li et al. 2020) featurise the example pairs based on their aligned attribute names $a_1, \ldots, a_k$ and values $v_{l_1} = r_l[a_1], \ldots, v_{l_k} = r_l[a_k], v_{r_1} = r_r[a_1], \ldots, v_{r_k} = r_r[a_k]$.

In contrast to classical entity matching, we address a more challenging yet practical setting, referred to as *zero-shot entity matching* (Wu et al. 2020). In particular, we approach the entity matching problem under the following restrictions:

Restriction 1 - Unseen target data: *A zero-shot matcher will have no access to labelled example pairs for the target relations* $\mathrm{R}_{\mathit{left}}$ *and* $\mathrm{R}_{\mathit{right}}$*, which means no training set is available for the unseen target data* $\mathcal{D}_{\mathit{target}}$ *.*

Restriction 2 - Lack of type information: *There is no column name or column type information accessible for the target relations* $\mathrm{R}_{\mathit{left}}$ *and* $\mathrm{R}_{\mathit{right}}$*. A zero-shot 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.*

Our proposed zero-shot EM setup is essential in scenarios requiring a high degree of automation, where it is unlikely or impractical to have a domain expert manually label training data. Prior work, such as *ZeroER* (Wu et al. 2020), addresses zero-shot entity matching under the constraint of no training data for target relations (Restriction 1). However, it still relies on column type information to select similarity functions, thus violating Restriction 2.

3 Approach

Following the approach in (Li et al. 2020), we treat EM as a sequence classification problem, where input records are serialized into an ordered textual sequence (see Appendix A.1 for an example of this serialization method). A classification model then predicts whether the records refer to the same real-world entity. We introduce the main idea of AnyMatch in the following and refer to Appendix A.2 for further details. Figure 2 illustrates the pipeline of using AnyMatch.

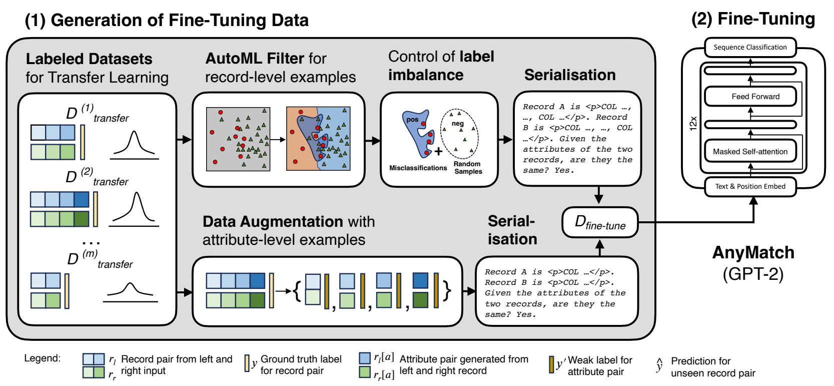
Zero-shot EM as a transfer learning problem. We aim for a matcher which can be applied to unseen target $\mathcal{D}_{\text{target}}$ originating from two relations R_{left} and $\mathrm{R}_{\text{right}}$, for which no labelled training data $\mathcal{D}_{\text{train}}$ is available. To address this, we leverage a small language model as a sequence classifier fine-tuned in a transfer learning setup. We select `GPT-2` (Radford et al. 2019) as the base model for its effective capability of contextual understanding. AnyMatch assumes that one can access a large set of m labelled datasets $\mathcal{D}^{(1)}_{\text{transfer}}, \cdots, \mathcal{D}^{(m)}_{\text{transfer}}$ from other relations (e.g., all publicly available academic benchmark datasets) to fine-tune a language model as the zero-shot matcher f.

Directly concatenating samples from all available datasets is prone to overfitting the model, thus special care is required to create high quality fine-tuning data. We design two data selection techniques to make sure that $\mathcal{D}_{\text{fine-tune}}$ contains examples that result in a generalisable matcher.

Boosting the matcher with difficult examples. A general principle in many EM systems is to categorise candidate pairs based on the difficulty of distinguishing between the represented entities. Unlikely matches can easily be pruned early via blocking functions (Papadakis et al. 2020). Moreover, recent studies (Mudgal et al. 2018; Leone et al. 2022; Papadakis et al. 2024) indicate that many entity matching benchmark datasets can already be solved using linear models.

These observations motivate us to think about EM datasets as a mix of record pairs that can easily classified by conventional machine learning models, such as linear models, and another set of pairs that require non-linearity to be solved by language models. We prioritise the inclusion of the latter, with more challenging examples be placed into the fine-tuning data $\mathcal{D}_{\text{fine-tune}}$ via an *AutoML filter*: we train an AutoML model on examples from a labelled dataset $\mathcal{D}^{(i)}_{\text{transfer}}$ and include samples misclassified by the AutoML model in $\mathcal{D}_{\text{fine-tune}}$. Such samples often highlight the boundaries or edge cases that the model struggles with, and including them can lead to more robust performance and higher transferability.

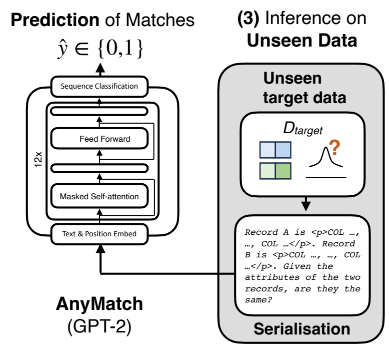
Augmentation with attribute-level samples. Even


(1) Generation of Fine-Tuning Data
Labeled Datasets for Transfer Learning
$D^{(1)}_{transfer}$
$D^{(2)}_{transfer}$
...
$D^{(m)}_{transfer}$
AutoML Filter for record-level examples
Control of label imbalance
pos
neg
Misclassifications
Random Samples
Serialisation
Data Augmentation with attribute-level examples
Serial-isation
$D_{fine-tune}$
(2) Fine-Tuning
Sequence Classification
Feed Forward
Masked Self-attention
Text & Position Embed
12x
AnyMatch
(GPT-2)
Legend:
r_l r_r Record pair from left and right input
y Ground truth label for record pair
$r_l[a]$ $r_r[a]$ Attribute pair generated from left and right record
y' Weak label for attribute pair
$\hat{y}$ Prediction for unseen record pair

Record A is <p>COL …, …, COL …</p>. Record B is <p>COL …, …, COL …</p>. Given the attributes of the two records, are they the same? Yes.

Record A is <p>COL …</p>. Record B is <p>COL …</p>. Given the attributes of the two records, are they the same? Yes.

(a) Fine-tuning on high quality data generated from $\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}$.



Record A is <p>COL …, …, COL …</p>. Record B is <p>COL …, …, COL …</p>. Given the attributes of the two records, are they the same?

(b) Inference on data $\mathcal{D}_{\text{target}}$.

Figure 2: High-level overview of AnyMatch – (1) We generate fine-tuning data from the available labelled datasets by applying several data selection techniques, and (2) fine-tune a language model as zero-shot entity matcher. (3) We use the resulting matcher for inference on unseen target data at deployment time.

though LLMs show promising potential for data wrangling (Narayan et al. 2022), there still remain fundamental structural mismatches in applying them to relational data (Badaro, Saeed, and Papotti 2023), e.g., that relational data has an additional column/attribute structure (which is not present in text) and that the relational model does not impose an order over these attributes. We accommodate for this structural mismatch with an augmentation approach that is inspired by how humans tackle the entity matching task: they will not immediately make decisions at the record level, but will use a combination of comparisons across different attributes.

In order to account for this observation, we generate *additional attribute-level examples* which only contain pairs of attribute values $(r_l[a], r_r[a])$ for each attribute a from a labeled record pair (r_l, r_r) in $\mathcal{D}^{(i)}_{\text{transfer}}$, and augment the fine-tuning data $\mathcal{D}_{\text{fine-tune}}$ with these additional samples.

4 Experimental Evaluation

We first assess the predictive quality of AnyMatch in comparison to thirteen baselines (details provided in Appendix A.4) on nine widely used EM datasets (details provided in Appendix A.3). Ablation results (detailed in Appendix A.5) further validate the design choices of our method. Additionally, we highlight the deployment cost benefits of our method compared to other large language model-based matchers.

4.1 Prediction Quality

We evaluate the zero-shot matching quality using a "leave-one-dataset-out" methodology: to test on an unseen target dataset, we use the remaining eight datasets as transfer data for training or prompting. In line with existing research, we report the F1 score for the predictions on the entity matching tasks.

The resulting F1 scores for the nine datasets are shown in Table 1, with the highest score for each dataset in bold and the second-highest score underlined. Scores for `Jellyfish` are shown in brackets where it was pretrained on that dataset, indicating a violation of the zero-shot setting. Results for `GPT-3` (Narayan et al. 2022) and `TableGPT` (Li et al. 2024) are taken from their original papers, as these models have been deprecated or are not publicly available.

`MatchGPT` (Peeters and Bizer 2023) with the proprietary trillion-parameter model `GPT-4` achieves the highest average F1 score of 86.36. We find that AnyMatch provides the second-highest overall performance with a mean F1 score of 81.96. This is remarkable since it outperforms `MatchGPT` with all open and commercial models (except for GPT-4), which have up to three orders of magnitude more parameters than AnyMatch. We find that our method provides the best performance on the three datasets `BEER`, `FOZA` and `ITAM` from diverse domains and the second-best performance on `ABT` and `DBGO`. However, for other datasets such as `AMGO` and `WAAM`, there is still a gap compared to `MatchGPT [GPT-4]`. We attribute this to the fact that these dataset employ very specific language to describe products, which lacks grammatically consistency. Another unknown factor in these results is the question whether the commercial models have already seen the (publicly available) benchmark datasets at training time. This is impossible to determine however, since the training data of these models is not disclosed.

As expected, the `StringSim` baseline gives the worst average performance of 40.21. `TableGPT` (which again applies models with hundreds of billions of parameters) also scores high on the datasets for which its authors provide numbers in (Li et al. 2024). AnyMatch also outperforms `Jellyfish` (Zhang et al. 2023) with a score of 81.96 compared to 77.83, even though the latter has seen data from six out the nine datasets at training time. The `Ditto` (Li et al. 2020) baseline (which we trained in a similar manner as our method and which has a similarly sized base model) also performs well with a score of 66.05 (and achieves on par performance with some of the larger models), but is more than 15 points behind our method. Unicorn (Tu et al. 2023),

Table 1: F1 scores for zero-shot EM. (Best score in bold, second-best score underlined, numbers in brackets indicate the model has seen data from a dataset at training time).

	#params (millions)	Unseen Target Dataset									
		ABT	AMGO	BEER	DBAC	DBGO	FOZA	ITAM	WAAM	WDC	Mean
StringSim	-	32.19	36.43	29.55	73.94	60.10	22.00	51.28	27.98	28.46	40.21
ZeroER	-	54.13	45.79	67.67	93.31	85.15	**100.00**	70.12	43.12	42.46	66.86
Ditto	110	53.94	50.28	73.68	95.22	87.79	69.84	72.97	39.87	50.86	66.05
Unicorn	145	89.45	55.70	90.32	94.33	**94.11**	90.91	72.22	63.57	47.28	77.54
MatchGPT [GPT-4o-mini]	8,000	89.67	65.24	66.67	92.07	86.59	93.02	72.73	83.15	79.91	81.01
Jellyfish	13,000	81.13	(59.72)	(80.00)	(98.40)	(92.65)	(97.67)	(82.61)	65.78	42.55	(77.83)
MatchGPT [Mixtral-8x7B]	56,000	79.02	31.65	70.91	87.63	66.15	88.23	62.13	50.56	42.04	64.26
MatchGPT [SOLAR]	70,000	85.04	19.01	71.23	88.60	52.05	89.37	61.20	65.32	56.52	65.37
MatchGPT [Beluga2]	70,000	83.51	42.01	76.34	92.43	72.68	82.17	54.36	63.46	54.97	69.10
GPT-3	175,000	n/a	54.3	78.6	93.5	64.6	87.2	65.9	60.6	n/a	n/a
TableGPT [GPT-3.5-text-davinci-002]	175,000	n/a	65.7	72.7	84.7	86.1	87.2	78.8	69.1	n/a	n/a
TableGPT [GPT-3.5-text-chat-davinci-002]	175,000	n/a	56.6	92.3	93.2	91.1	**100.0**	86.2	67.8	n/a	n/a
MatchGPT [GPT-3.5-Turbo03]	175,000	74.32	57.91	78.78	88.73	79.15	82.35	56.10	64.26	76.51	73.12
MatchGPT [GPT-3.5-Turbo06]	175,000	82.30	44.06	70.00	93.28	78.22	93.02	57.57	68.77	60.62	71.98
MatchGPT [GPT-4]	1,760,000	**94.40**	**74.91**	69.57	**95.60**	87.22	97.67	82.35	**89.67**	**85.83**	**86.36**
AnyMatch [GPT-2]	124	86.05	55.08	**96.55**	93.61	90.59	**100.00**	**90.91**	61.51	63.31	81.96

the successor to Ditto, aims to achieve zero-shot functionality using a mixture-of-experts architecture. However, it still falls short of our method by 4% in performance. The parameterless ZeroER (Wu et al. 2020) also scores high (and provides the best performance on DBAC and FOZA) and even outperforms Ditto. However, it still lags more than 14 points behind our method.

In summary, we find that AnyMatch outperforms all baselines except for one and provides an average matching performance that is within 4.4% percent of the best observed performance from MatchGPT [GPT-4] that is four orders of magnitude more parameters than AnyMatch.

4.2 Deployment Cost for Inference

Next, we assess the deployment cost of AnyMatch in comparison to other zero-shot matchers. As previously demonstrated, matchers that rely on large language models are the primary competitors among existing methods. Since the Jellyfish (Zhang et al. 2023) model does not strictly adhere to the zero-shot restrictions, GPT-3 (Narayan et al. 2022) is deprecated, and TableGPT (Li et al. 2024) is not open-sourced, our comparison focuses on the inference costs associated with MatchGPT solutions. Given that MatchGPT utilizes both open-weight and large proprietary models, different strategies are needed to quantify these costs.

For AnyMatch and the open-weight MatchGPT models, including Mixtral, SOLAR, and Beluga2, we use throughput as a proxy for computational efficiency, measured in tokens per second (Borzunov et al. 2024), to quantify deployment costs. To ensure a fair comparison, all models are deployed on the same hardware configuration; we record the maximum batch size each model can accommodate and then interpolate the throughput values in tokens per second. Using this throughput, we calculate the deployment cost per 1,000 tokens based on prices from cloud platforms like Amazon Web Services or AI-specific platforms such as together.ai. Unfortunately, we are unable to measure the throughput of commercial models like GPT-4 due to their deployment behind proprietary APIs. The underlying hardware configurations are unknown, and these models likely operate in multi-tenant settings. Therefore, we use their respective API costs for comparison.

We focus on the trade-off between the prediction quality of these approaches and their deployment cost. For that. we plot the average F1 score achieved versus the estimated cost for 1,000 tokens from the analysis in Figure 1. Note that we only include methods whose models are not deprecated and which satisfy our zero-shot constraints. More details about the setup and inference throughput can be found in Appendix A.7.

MatchGPT [GPT-4] had the highest average F1-score of 86.36 in the zero-shot setting. Our proposed model AnyMatch reached the second highest F1 score of 81.96, with only a 4.4% difference, even though it has four orders of magnitude less parameters (124 million vs 1.76 trillion). The slight performance benefit of MatchGPT [GPT-4] comes at a drastic price increase though, as it has a 3,899 times higher inference cost than AnyMatch. These numbers showcase the attractive trade-off between prediction quality and deployment cost offered by AnyMatch, which suggest that AnyMatch should be preferred in scenarios where cost, scale and speed have priority over peak prediction quality.

5 Conclusion

We revisited the zero-shot EM problem with AnyMatch, a small language model fine-tuned through transfer learning, introducing several novel data selection techniques to generate high-quality fine-tuning data. Our method was comprehensively compared to thirteen baselines across nine datasets. AnyMatch achieves competitive prediction quality and offers substantial deployment cost advantages over recent EM approaches that rely on trillion-parameter LLMs. Besides, we also observe that these data selection strategies, effective for AnyMatch, become less impactful as model size is large enough (see Appendix A.8 for details), which motivates future investigation.

Acknowledgements. This research was supported by the University of Amsterdam Data Science Center.

References

Abdelaal, M.; Hammacher, C.; and Schoening, H. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. *Proceedings of the VLDB Endowment (PVLDB)*.

Abedjan, Z.; Chu, X.; Deng, D.; Fernandez, R. C.; Ilyas, I. F.; Ouzzani, M.; Papotti, P.; Stonebraker, M.; and Tang, N. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12): 993–1004.

Badaro, G.; Saeed, M.; and Papotti, P. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics*, 11: 227–249.

Borzunov, A.; Ryabinin, M.; Chumachenko, A.; Baranchuk, D.; Dettmers, T.; Belkada, Y.; Samygin, P.; and Raffel, C. A. 2024. Distributed inference and fine-tuning of large language models over the internet. *Advances in Neural Information Processing Systems*, 36.

Chen, R.; Shen, Y.; and Zhang, D. 2021. GNEM: a generic one-to-set neural entity matching framework. In *Proceedings of the Web Conference 2021*, 1686–1694.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Doan, A.; Konda, P.; Suganthan GC, P.; Govind, Y.; Paulsen, D.; Chandrasekhar, K.; Martinkus, P.; and Christie, M. 2020. Magellan: toward building ecosystems of entity matching solutions. *Communications of the ACM*, 63(8): 83–91.

Erickson, N.; Mueller, J.; Shirkov, A.; Zhang, H.; Larroy, P.; Li, M.; and Smola, A. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*.

Fan, G.; Shraga, R.; and Miller, R. J. 2024. Gen-T: Table Reclamation in Data Lakes. *arXiv preprint arXiv:2403.14128*.

Fernandez, R. C.; Elmore, A. J.; Franklin, M. J.; Krishnan, S.; and Tan, C. 2023. How large language models will disrupt data management. *Proceedings of the VLDB Endowment*, 16(11): 3302–3309.

Fu, C.; Han, X.; He, J.; and Sun, L. 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.

Gao, Y.; Huang, S.; and Parameswaran, A. 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.

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

Hynes, N.; Sculley, D.; and Terry, M. 2017. The Data Linter: Lightweight Automated Sanity Checking for ML Data Sets. *Machine Learning Systems workshop at NeurIPS*.

Leone, M.; Huber, S.; Arora, A.; García-Durán, A.; and West, R. 2022. A critical re-evaluation of neural methods for entity alignment. *Proceedings of the VLDB Endowment*, 15(8): 1712–1725.

Li, P.; He, Y.; Yashar, D.; Cui, W.; Ge, S.; Zhang, H.; Rifinski Fainman, D.; Zhang, D.; and Chaudhuri, S. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proceedings of the ACM on Management of Data*, 2(3): 1–28.

Li, P.; Rao, X.; Blase, J.; Zhang, Y.; Chu, X.; and Zhang, C. 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)*, 13–24. IEEE.

Li, Y.; Li, J.; Suhara, Y.; Doan, A.; and Tan, W.-C. 2020. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*.

Liu, C.; Russo, M.; Cafarella, M.; Cao, L.; Chen, P. B.; Chen, Z.; Franklin, M.; Kraska, T.; Madden, S.; and Vitagliano, G. 2024. A Declarative System for Optimizing AI Workloads. *arXiv preprint arXiv:2405.14696*.

Ma, J.; Zhao, Z.; Yi, X.; Chen, J.; Hong, L.; and Chi, E. H. 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.

Mudgal, S.; Li, H.; Rekatsinas, T.; Doan, A.; Park, Y.; Krishnan, G.; Deep, R.; Arcaute, E.; and Raghavendra, V. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 international conference on management of data*, 19–34.

Narayan, A.; et al. 2022. Can Foundation Models Wrangle Your Data? *PVLDB*.

Papadakis, G.; Kirielle, N.; Christen, P.; and Palpanas, T. 2024. A critical re-evaluation of benchmark datasets for (deep) learning-based matching algorithms. *ICDE*.

Papadakis, G.; Skoutas, D.; Thanos, E.; and Palpanas, T. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2): 1–42.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Peeters, R.; and Bizer, C. 2023. Entity matching using large language models. *arXiv preprint arXiv:2310.11244*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8): 9.

Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; and Liu, P. J. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67.

Services, A. W. 2022. AWS Glue. https://aws.amazon.com/glue/.

Stonebraker, M.; Ilyas, I. F.; et al. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.*, 41(2): 3–9.

Tu, J.; Fan, J.; Tang, N.; Wang, P.; Li, G.; Du, X.; Jia, X.; and Gao, S. 2023. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data*, 1(1): 1–26.

Vos, D.; Döhmen, T.; and Schelter, S. 2022. Towards parameter-efficient automation of data wrangling tasks with prefix-tuning. In *NeurIPS 2022 First Table Representation Workshop*.

Wang, J.; Li, Y.; Hirota, W.; and Kandogan, E. 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.

Wang, J.; Wang, J.; Athiwaratkun, B.; Zhang, C.; and Zou, J. 2024. Mixture-of-Agents Enhances Large Language Model Capabilities. *arXiv preprint arXiv:2406.04692*.

Wu, R.; Chaba, S.; Sawlani, S.; Chu, X.; and Thirumuruganathan, S. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 1149–1164.

Zhang, H.; Dong, Y.; Xiao, C.; and Oyamada, M. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *arXiv preprint arXiv:2312.01678*.

Zhang, Y.; and Ives, Z. G. 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.

Zhang, Z.; Groth, P.; Calixto, I.; and Schelter, S. 2024. Directions Towards Efficient and Automated Data Wrangling with Large Language Models. *Database and Machine Learning workshop at ICDE*.

A Appendix

A.1 Example of Serialisation

ANYMATCH serializes two records into a prompt format that is independent of the data domain and does not require any schema information such as column names or types. For instance, given two records from the music domain representing two different songs by the artist "David Guetta" :

```
[{song_name: "I'm a Machine", musician:
↪  "David Guetta",
 price: "$1.29", release: 2011},
{title: "Night Of Your Life", artist: "David
↪  Guetta",
 price: "$1.29", year: 2011}]
```

the serialized prompt fed to the language model would appear as follows:

```
Record A is <p>COL I'm a Machine, COL David
↪  Guetta,
COL $1.29, COL 2011</p>. Record B is <p>COL
↪  Night Of Your
Life, COL David Guetta, COL $1.29, COL
↪  2011</p>. Given the
attributes of the two records, are they the
↪  same?
```

A.2 Details of Approach

We now detail the concrete algorithms to turn the set of labelled transfer datasets $\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}$ into the data $\mathcal{D}_{\text{fine-tune}}$ which we use for fine-tuning. The function FINETUNE_ANYMATCH in Algorithm 1 shows the high-level procedure as outlined previously: we first generate record-level examples (Algorithm 2) which are then augmented with attribute-level examples (Algorithm 3), which we finally use to fine-tune our base model `GPT-2`.

Record-level sample generation with AutoML selection. We generate record-level samples independently from each dataset $\mathcal{D}_{\text{transfer}} \in \mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}$ as described in Algorithm 2. On a high-level, we aim to resample each dataset $\mathcal{D}_{\text{transfer}}$ to have a ratio of twice as much non-matching as matching record pairs. Furthermore, we focus on identifying and including difficult matching pairs in the fine-tuning data generated from $\mathcal{D}_{\text{transfer}}$.

We generate n_r samples from each dataset $\mathcal{D}_{\text{transfer}}$ as follows. First, note that n_r is a hyperparameter and the we use the full dataset $\mathcal{D}_{\text{transfer}}$ if it has less pairs (Line 19). We detect difficult matching pairs by training an AutoML classifier f_{auto} on $\mathcal{D}_{\text{transfer}}$ and identify the set D^{+}_{wrong} of matching record pairs that were misclassified by f_{auto} as false negatives (Lines 4-5). We leverage these misclassified samples to create the set of matching pairs to use for fine-tuning D^{+}, by sampling $\frac{1}{3}n_r$ difficult matching pairs from D^{+}_{wrong} (and additionally include correctly classified pairs as well if there are not enough difficult pairs, Lines 6-15). Next, we randomly sample twice as many non-matching pairs from $\mathcal{D}_{\text{transfer}}$ to form the set of non-matching pairs D^{-}. Finally, we combine D^{+} and D^{-} to form the set of pairs $D^{\pm}$ and turn each retained labelled record pair (r_l, r_r, y) from $D^{\pm}$ into two instances based on our serialisation format (Lines 20-24). We generate a sample based on the value pairs (V_l, V_r) and a second sample where the order of the value pairs is flipped: (V_r, V_l). At this level, all attribute values V_l from the left record r_l and V_r from the right record r_r are included in the serialisation.

Algorithm 1: Fine-tuning ANYMATCH via transfer learning.

```
1 function FINETUNE_ANYMATCH($\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}, n_a, n_r$)
2   $\mathcal{D}_{\text{fine-tune}} \leftarrow \emptyset$
    – Generate record-level samples
3   for $i \in 1 \ldots m$ do
4     $S_{\text{record}} \leftarrow$ RECORDLEVEL($\mathcal{D}^{(i)}_{\text{transfer}}, n_r$)
5     $\mathcal{D}_{\text{fine-tune}} \leftarrow \mathcal{D}_{\text{fine-tune}} \cup S_{\text{record}}$
    – Generate attribute-level samples
6   $S_{\text{attribute}} \leftarrow$ ATTRIBUTELEVEL($\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}, n_a$)
7   $\mathcal{D}_{\text{fine-tune}} \leftarrow \mathcal{D}_{\text{fine-tune}} \cup S_{\text{attribute}}$
    – Fine-tune a small language model for EM
8   $f \leftarrow$ fine-tune GPT-2 on $\mathcal{D}_{\text{fine-tune}}$
9   return $f$
```

Augmentation with attribute-level samples. In addition to the record-level samples, we also generate attribute-level samples for fine-tuning. We detail their generation process in Algorithm 3. The goal of this function is to generate samples[1] for all attributes A occurring in any of the datasets $\mathcal{D}^{(0)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}$. Therefore, the data access pattern is different than before. Instead of processing each dataset $D^{(j)}_{\text{transfer}}$ independently, we now group the data by attribute (Lines 5-11) and process each record group G containing all labelled pairs of values of a particular attribute A_i independently. We balance each group G to contain positively and negatively labelled attribute pairs in equal proportions and make sure that we use at most n_a pairs per attribute (Lines 12-19). Note that the threshold n_a is a hyperparameter again. Recall that we use a ratio of one to two for record-level samples, but we do not apply the same ratio for attribute-level samples. This is because attribute-level samples are generated using simple heuristic rules without ground-truth labels on attribute-level matches. Therefore, we use a balanced ratio between positive and negative samples to minimise inductive bias. Finally, we turn each retained labelled attribute value pair into a sample based on our serialisation format, using only a single attribute value v_l from the left record r_l and the corresponding single attribute value v_r from the right record r_r (Line 21).

A.3 Datasets

We experiment on nine benchmark datasets detailed in Table 2, which are commonly used in entity matching research (Li et al.

[1]Note that this requires us to have access to attribute names at data generation time, however the actual attribute names are not part of the generated fine-tuning data and thereby not observed by our model.

Algorithm 2: Generating record-level fine-tuning samples.

```
1  function RECORDLEVEL($\mathcal{D}_{\text{transfer}}, n_r$)
2    $S_{\text{record}} \leftarrow \emptyset$
3    if $|\mathcal{D}_{\text{transfer}}| > n_r$ then
       – AutoML filter to select difficult examples
4      $f_{\text{auto}} \leftarrow$ train AutoML classifier on $\mathcal{D}_{\text{transfer}}$
5      $D^+_{\text{wrong}} \leftarrow$ matching pairs misclassified by $f_{\text{auto}}$
       – Downsampling to control label imbalance
6      $n^+_{\text{wrong}} \leftarrow |D^+_{\text{wrong}}|$
7      $n^+_D \leftarrow$ number of matching pairs in $\mathcal{D}_{\text{transfer}}$
8      $n_p \leftarrow \frac{1}{3} n_r$
9      $n^+ \leftarrow \min(n^+_D, n_p)$
10     if $n^+_{\text{wrong}} \geq n^+$ then
11       $D^+ \leftarrow$ sample $n^+$ pairs from $D^+_{\text{wrong}}$
12     else
13       $D^+_{\text{corr}} \leftarrow$ matching pairs correctly classified by $f_{\text{auto}}$
14       $D^+_{\text{corr}} \leftarrow$ sample $(n^+ - n^+_{\text{wrong}})$ pairs from $D^+_{\text{corr}}$
15       $D^+ \leftarrow D^+_{\text{wrong}} \cup D^+_{\text{corr}}$
16     $D^- \leftarrow$ sample $2n^+$ non-matching pairs from $\mathcal{D}_{\text{transfer}}$
17     $D^{\pm} \leftarrow D^+ \cup D^-$
18   else
       – No filtering for tiny datasets
19     $D^{\pm} \leftarrow \mathcal{D}_{\text{transfer}}$
     – Generation of serialised samples
20   for $(r_l, r_r, y) \in D^{\pm}$ do
21     $V_l \leftarrow$ ENUMERATE_ATTRIBUTE_VALUES($r_l$)
22     $V_r \leftarrow$ ENUMERATE_ATTRIBUTE_VALUES($r_r$)
23     $S_{\text{record}} \leftarrow S_{\text{record}} \cup$ SERIALIZE($V_l, V_r, y$)
       – Include "flipped" sample
24     $S_{\text{record}} \leftarrow S_{\text{record}} \cup$ SERIALIZE($V_r, V_l, y$)
25   return $S_{\text{record}}$
```

2020; Narayan et al. 2022; Peeters and Bizer 2023). . These datasets cover diverse domains and vary in sample sizes.

Our most important baseline (with the highest performance for zero-shot EM) is `MatchGPT` (Peeters and Bizer 2023), which leverages commercial LLM APIs. Due to the high costs of these APIs, the `MatchGPT` study down-samples a test set if it exceeds 1,250 samples (and reduces it to at most 250 positive samples and 1,000 negative samples). We adapt their methodology and consult `MatchGPT`'s code and data repository[2] to make sure that our test sets contain exactly the same record pairs.

Acronym	Dataset	Domain	#Samples
ABT	Abt-Buy	web products	8,865
AMGO	Amazon-Google	software	11,460
BEER	Beer	food	450
DBAC	DBLP-ACM	citation	12,363
DBGO	DBLP-Google	citation	28,707
FOZA	Fodors-Zagats	food	946
ITAM	iTunes-Amazon	music	539
WAAM	Walmart-Amazon	electronics	10,242
WDC	WDC	web products	6,239

Table 2: Benchmark datasets from (Narayan et al. 2022) and (Peeters and Bizer 2023) with their corresponding domain.

Algorithm 3: Generating attribute-level fine-tuning samples.

```
1  function ATTRIBUTELEVEL($\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}, n_a$)
2    $S_{\text{attribute}} \leftarrow \emptyset$
     – Identify all possible attributes
3    $A \leftarrow$ all attributes appearing in $\mathcal{D}^{(1)}_{\text{transfer}}, \ldots, \mathcal{D}^{(m)}_{\text{transfer}}$
     – Collect all attribute pairs
4    for attribute $a_i \in A$ do
5      $G \leftarrow \emptyset$
6      for $j \in 1 \ldots m$ do
7        if $a_i$ in attributes of $\mathcal{D}^{(j)}_{\text{transfer}}$ then
8          for $(r_l, r_r, y) \in \mathcal{D}^{(j)}_{\text{transfer}}$ do
9            $v_l \leftarrow r_l[a_i]$
10           $v_r \leftarrow r_r[a_i]$
11           $G \leftarrow G \cup (v_l, v_r, y)$
       – Balance the pairs per attribute
12     $n^+ \leftarrow$ number of matching pairs in $G$
13     $n^- \leftarrow$ number of non-matching pairs in $G$
14     $n_{\text{balance}} \leftarrow \min(|n^+|, |n^-|)$
15     $G^+ \leftarrow n_{\text{balance}}$ matching pairs sampled from $G$
16     $G^- \leftarrow n_{\text{balance}}$ non-matching pairs sampled from $G$
17     $G^{\pm} \leftarrow P^+ \cup P^-$
       – Down-sample the pairs per attribute
18     if $|G^{\pm}| > n_{\text{a}}$ then
19       $G^{\pm} \leftarrow$ randomly sample $n_{\text{a}}$ pairs from $P^{\pm}$
       – Generate attribute-level samples
20     for $(v_l, v_r, y) \in P^{\pm}$ do
21       $S_{\text{attribute}} \leftarrow S_{\text{attribute}} \cup$ SERIALIZE($v_l, v_r, y$)
22   return $S_{\text{attribute}}$
```

A.4 Compared Baselines

In this section, we will introduce the baselines we used to compare with ANYMATCH.

StringSim – a trivial baseline method, which serialises both tuples to compare by casting each column to a string and concatenating the values with a comma separator. Next, 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.

ZeroER (Wu et al. 2020) is a parameter-less matching method, explicitly designed for the zero-shot case without any training data for the target dataset. 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. In contrast to our approach, there are several drawbacks though: the method requires information about the types of the columns and a decision on the similarity functions to use, is only applicable in a batch setting and cannot match single record pairs in isolation (which for example makes debugging false predictions difficult),

[2]https://github.com/wbsg-uni-mannheim/MatchGPT/tree/main/LLMForEM

and relies on distributional assumptions which may not hold on every dataset.

Ditto (Li et al. 2020) is a state-of-the-art entity matching approach, based on fine-tuning a Bert encoder (Devlin et al. 2018). Ditto injects domain knowledge into the data during serialisation and augments the training data to enhance the model's ability to differentiate between challenging entity pairs. This process includes actions such as dropping columns and editing spans of tokens. Since Ditto models do not rely on a hard-coded schema during training, they can also be applied to unseen target datasets with a different schema in a zero-shot setting. In a similar direction, (Zhang et al. 2024) recently proposed a vision to leverage a selection of LoRA-tuned domain-specific models for entity matching.

Similarly, Tu et al. proposed *Unicorn* (Tu et al. 2023), a unified multi-tasking model designed to support various matching tasks in data integration. Unicorn adopts the core principle of multi-task learning by using a mixture of expert architecture (Ma et al. 2018), aiming to learn specialised expert models for different tasks. The primary workflow of Unicorn can be summarised as follows: first, data samples from different tasks are 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 final 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.

Jellyfish (Zhang et al. 2023) is a general LLM-based approach targeting four data preprocessing tasks (including entity matching). Jellyfish leverages two LLaMA2-13B models, which are fine-tuned in an instruction tuning fashion. The corresponding 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 tackle zero-shot data preparation scenarios on unseen datasets.

Narayan et al. (Narayan et al. 2022) showed that prompting large commercial LLMs with serialised records and carefully chosen few-shot examples can lead to competitive matching performance, referred as *GPT-3* in our paper.

MatchGPT (Peeters and Bizer 2023) enhances the chosen prompts and evaluates a wide variety of base models and prompt formats for both zero-shot and few-shot entity matching. *TableGPT* (Li et al. 2024) applies a similar approach, but enhances the LLMs with "table fine-tuning" to teach them various data preparation tasks. This approach is designed for both zero-shot and few-shot scenarios.

A.5 Ablation

We conduct ablation study on the base model selection and data generation strategies. We summarise the tested variants together with the corresponding results for the individual datasets in Table 3. In addition, we report the performance delta (the reduction in average F1 score) for all tested variants, in comparison to the proposed design of AnyMatch.

We first evaluate the impact of replacing `GPT-2` in AnyMatch with different similarly sized alternative models. In particular, we evaluate Google's `T5` (Raffel et al. 2020) model with an encoder-decoder architecture and the encoder-only model `Bert` (Devlin et al. 2018), which is for example also used by (Li et al. 2020). As detailed in Table 3, both alternative models result in a decrease of the overall F1 score: using `T5` results in a decrease of 2.38%, while using `Bert` leads to the drastic decrease of 9.04% in average F1 score. We attribute the significant decrease in predictive quality when using `Bert` to the following factors: `Bert` encodes the input into a vectorised representation, to which a prediction head is subsequently appended for making predictions. However, in our approach, we incorporate a task description into the input sequence, accounting for roughly 5% of the total input, which potentially negatively influences the downstream classification. Note while using a different serialization method for the `Bert` model might improve performance, this is not the focus of this ablation study.

Next, we aim to validate the choice of the different techniques used during the generation of training data for AnyMatch. As discussed, our main model uses difficult pair selection via AutoML (`automl`) for record-level instances, augments the data with "flipped" record pairs (`flip`) and mixes attribute-level training instances into the training data (`attr_mix`). In this experiment, we evaluate the following reduced variants:

- `(automl, attr_mix)` – This variant does not include flipped record pairs.
- `(attr_mix)` – This variant does not use our AutoML-based approach to select difficult matching examples and does not include flipped record pairs.
- `(attr_seq)` – This variant also does not use our AutoML-based approach to select difficult matching examples and does not include flipped record pairs. Additionally, it does not mix the attribute-level training instances with the record-level training instances. Instead, it uses sequential training to fine-tune the model on the attribute-level pairs before continuing the fine-tuning on record-level examples.
- `()` – This variant neither uses the AutoML-based approach to select difficult matching examples nor any attribute-level training examples or flipped record pairs.

The results Table 3 validate the benefits of our training data generation techniques, as we find that the removal of any of our proposed techniques results in a performance decrease. Removing the flipped records pairs in `(automl, attr_mix)` leads to a decrease of 0.72%, additionally removing the selection of difficult examples via AutoML in `(attr_mix)` further decreases the performance by 1.62%. Removing or not mixing in the attribute level instances (e.g., the variants `(attr_seq)` and `()`) leads to a performance loss of more than 3%.

A.6 Implementation Details & Hyperparameters

We implement AnyMatch based on the existing `GPT2Tokenizer` and `GPT2ForSequenceClassification` classes in PyTorch (Paszke et al. 2019) for our chosen base model `GPT-2` (Radford et al. 2019) from the `transformers` library.

We implement the AutoML filter, which selects difficult training examples, based on the `TabularPredictor` from Amazon's *AutoGluon* library (Erickson et al. 2020). When filtering individual datasets, we convert them into a `TabularDataset` instance and subsequently construct a `TabularPredictor` by specifying the name of the label column that indicates matches. We do not constrain the time budget for the AutoML training (e.g., we do not set a `time_limit` argument in the `fit()` method), but observe that the fitting procedure can be completed within minutes for the datasets we are using.

We use a fixed set of hyperparameters across all evaluations. We set the number of record-level instances to generate per dataset $n_r = 1,200$ and the number of attribute-level instances to generate per attribute $n_a = 600$. When we fine-tune AnyMatch, we use a learning rate of 0.00002 and always choose the largest batch size that fits into the memory of the underlying GPU.

	Unseen Target Dataset									
	ABT	AMGO	BEER	DBAC	DBGO	FOZA	ITAM	WAAM	WDC	Mean
AnyMatch (main model)	86.05	55.08	96.55	93.61	90.59	100.00	90.91	61.51	63.31	81.96
Choice of base model – main model uses GPT2										
(GPT2) → (T5)	85.45	56.12	90.32	87.49	84.76	97.67	91.32	60.47	62.57	79.57 ($-\Delta 2.38$)
(GPT2) → (BERT)	73.82	51.28	87.46	89.72	86.54	89.72	72.65	43.58	61.42	72.91 ($-\Delta 9.04$)
Training data generation strategy — main model uses (automl, flip, attr_mix)										
(automl, flip, attr_mix) → (automl, attr_mix)	84.01	49.50	96.55	92.54	90.12	100.00	89.79	66.16	62.35	81.24 ($-\Delta 0.72$)
(automl, flip, attr_mix) → (attr_mix)	82.27	56.64	90.32	94.82	89.98	97.67	87.72	64.50	59.11	80.34 ($-\Delta 1.62$)
(automl, flip, attr_mix) → (attr_seq)	84.03	58.65	84.85	95.18	90.26	91.67	88.89	57.05	58.77	78.82 ($-\Delta 3.14$)
(automl, flip, attr_mix) → ()	82.82	52.91	87.50	93.79	89.58	97.67	90.91	55.71	58.05	78.77 ($-\Delta 3.19$)

Table 3: F1 scores from our ablation study for the choice of base model and the training data generation strategies of ANYMATCH. Altering any of our design choices leads to a performance decrease of up to 9.04 percent.

A.7 Deployment Cost

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.

Model	Used by	#params (millions)	RAM (GB)	batch size	Throughput (tokens/s)
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
GPT-2	ANYMATCH	124	0.26	8,192	**693,999**

Table 4: Throughput in tokens/s on a machine with 4xA100 (40GB) GPUs for different open-weight LLMs employed by Jellyfish and MatchGPT and our proposed model ANYMATCH. Due to the small number of parameters in ANYMATCH, its throughput is 25x higher than the throughput of Jellyfish and up to 922x higher than the throughput for the large models employed by MatchGPT. Note that ANYMATCH outperforms these models in terms of prediction quality as well, despite its drastically smaller number of parameters.

We list the required memory per model, the corresponding maximum usable batch size and the achieved throughput in Table 4. Both GPT-2 used by ANYMATCH and Llama2-13B used by Jellyfish 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. We observe that ANYMATCH has an up to 922x times higher throughput than MatchGPT with its large models and a 25x times higher throughput than Jellyfish. We attribute the low throughput numbers for MatchGPT to the high memory requirements, which force the use of model parallelism over multiple GPUs. The latter is detrimental for throughput since the model activation must be copied over to the memory of other GPUs. MatchGPT can only use small batch sizes (16-32) and achieves a low throughput in the range of 752 to 2,108 tokens per second only. In contrast, the Llama2-13B model used by Jellyfish fits into the memory of a single GPU, works with a higher batch size of 128, and achieves a throughput of over 26 thousand tokens/s. Our proposed method ANYMATCH can use a very large batch size of 8,192, since its base model GPT-2 has a comparably small parameter size of 124M, which only requires about 260M of GPU RAM for the model weights. This is two orders of magnitude less memory than Jellyfish and MatchGPT. As a result, ANYMATCH achieves a throughput of over 690 thousand tokens/s in this experiment, which is 25x higher than the throughput of ANYMATCH and between 329x to 922x higher than the throughput of MatchGPT with the large open-weight models. These throughput differences of up to two orders of magnitude are a clear indication of the performance benefits of leveraging a small language model for zero-shot entity matching.

We lookup the costs for the commercial models from OpenAI at https://openai.com/api/pricing/. As of November 2024, batch inference with the GPT-4 model costs $0.015 per 1,000 tokens and inference with GPT-3.5-turbo-0613 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, which only generates a single output.

We estimate the cost for ANYMATCH 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). We use the cost for a p4d.24xlarge instance[3] 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 June 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

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

same type of GPU (only twice the amount), we can extrapolate our throughput numbers from **??** 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`[4] 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).

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-06]`	$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`
AnyMatch	$0.0000038	8x on `p4d.24xlarge`

Table 5: 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. AnyMatch offers the lowest cost and three orders of magnitude cheaper than `MatchGPT` with the commercial `GPT-4` model.

We list the resulting costs per method and model in descending order in Table 5 . 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. We encounter the highest costs for `MatchGPT` with `GPT-4` (which gave the highest prediction quality). These costs are an order of magnitude higher than the cost for hosting `MatchGPT` with the `SOLAR` or `Beluga2` models or even using the older commercial model `GPT-3.5-turbo-06`. Using the smaller `Mixtral-8x7B` model is 23x cheaper than `GPT-4`, while `Jellyfish` (with a 13 billion parameter model) is already several orders of magnitude cheaper than `GPT-4`. AnyMatch is by far the cheapest model in this comparison, which is expected due to its low memory footprint and high throughout. It outperforms `GPT-4` by a factor of 3,899x in price.

Model	#Params (millions)	Mean F1	Cost for 1K tokens
GPT-2 w. DS	124	81.96	0.0000038
GPT-2 w/o DS	124	78.77	0.0000038
Llama3.2-1B w. DS	1,300	86.64	0.0001
Llama3.2-1B w/o DS	1,300	87.06	0.0001

Table 6: The mean F1 score for fine-tuning a larger language model, both with and without the data selection (DS) strategies used by AnyMatch (referred to as “GPT-2 w. DS”).

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

A.8 Fine-tune a Larger Language Model

We also applied the same fine-tuning approach to a larger language model, Llama3.2-1B, with 1.3 billion parameters—ten times larger than the GPT-2 model used in AnyMatch. However, when fine-tuning Llama3.2-1B, the data selection strategies were no longer beneficial, as evidenced in table 6. This finding suggests that smaller models benefit more from carefully designed data selection strategies.