Dynamic Selection of Value Matching Methods

Roque Lopez
New York University
New York, USA
rl3725@nyu.edu

Thiago Viegas
New York University
New York, USA
tjv235@nyu.edu

Amory Gao
New York University
New York, USA
axg212@nyu.edu

Remo Shen
New York University
New York, USA
ys6345@nyu.edu

ABSTRACT

Value matching is a core operation in data integration, yet existing approaches struggle to balance accuracy and computational efficiency. Transformation-driven systems rely heavily on handcrafted transformation pipelines, while model-centric methods achieve strong accuracy at the expense of substantial runtime and resource overhead. We propose a reinforcement learning (RL) framework that addresses these limitations by learning to select the most promising primitive from a set of value matching methods, using simple hints and metadata from the source and target values. At each step, the agent applies the chosen primitive, compares its prediction to the ground truth, and receives a reward that accounts for both performance and the primitive's cost. This enables the agent to learn policies that adapt matching strategies to the characteristics of each dataset. Our results show that the proposed approach effectively integrates diverse matching methods, offering a context-adaptive solution for value matching. We also compare our method to other well-known value matching systems, achieving better or comparable performance.

Source Code Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/cucumberpeel/thematchmakers.

1 INTRODUCTION

Reconciling values that refer to the same entity is a core requirement in data integration, data cleaning, and many machine learning workflows. Unifying heterogeneous datasets not only improves the quality and consistency of the underlying data but can also enhance model performance and lead to deeper, more reliable insights.

In practice, however, real-world data integration scenarios often involve columns containing heterogeneous representations of the same underlying entity—for example, "USA", "U.S.", "United States" and "United States of America" all refer to the same concept. Because of this variability, straightforward matching techniques—such as equi-joins that rely on exact string equality—often fail to identify semantically equivalent values. These inconsistencies arise across many domains, including geographic names, organization titles, biomedical vocabularies, and product identifiers, and they lead to low recall in realistic data integration pipelines. A wide range of semantic join techniques have been proposed to address the limitations of equi-joins. These techniques rely on different strategies to match the values in each column—a task commonly referred to as value matching. Classical approaches rely on lexical similarity[13], edit distance[3], or predefined transformation rules[19], which handle only shallow syntactic variation and fail to capture heterogeneous value formats commonly found in practice. Learning-based methods [7, 14] improve robustness but still depend on global similarity thresholds or a single transformation strategy, making them brittle across domains where different rows often require different normalization patterns. Recent transformation-search and mapping techniques [4] further highlight this limitation: most systems assume a small set of transformations or search for a single rule that must apply to all rows, leading to poor coverage or high runtime under increasing heterogeneity.

Emerging LLM-based solutions [5] provide stronger semantic reasoning but typically rely on uniform formats, curated examples, or expensive prompting procedures, and they offer limited guidance on selecting the appropriate reasoning or transformation path for diverse input values. Collectively, these constraints highlight a missing capability in current systems: a general-purpose approach that can adaptively choose the most suitable value-matching strategy—syntactic, semantic, or reasoning driven—according to the characteristics of the data, rather than relying on a single fixed method.

Existing approaches typically rely on a single method or primitive and thus suffer from complementary limitations. Rule-based methods capture only surface-level variations; embedding-based similarity degrades under distribution shift or ambiguous values; and LLM-driven techniques, while semantically expressive, remain too costly and unstable to apply uniformly at scale. These shortcomings highlight that real-world tabular heterogeneity spans multiple dimensions, making it infeasible for any single matching strategy to perform reliably across all data conditions.

This paper argues that an effective value matching must therefore adaptively combine the strengths of different primitives and select the most suitable one based on the characteristics of the input columns. We propose a multi-primitive value matching operator equipped with a reinforcement-learning (RL) selector that dynamically chooses among syntactic, embedding-based, and reasoning-driven strategies for each column pair. The selector observes signals such as the similarity between values and the semantic types of the columns, and it learns policies that maximize matching accuracy under computational constraints, drawing on recent advances in reinforcement learning for data-intensive systems.[8, 20].

Our contributions are as follows:

Unified Framework. We design a value matching operator that integrates syntactic, semantic, and reasoning-based matching primitives within a single, extensible architecture.

RL-Based Selector. We employ reinforcement learning to guide operator selection, balancing the performance of each primitive against its associated computational cost.

Empirical Evaluation. Through experiments on heterogeneous real-world datasets, we demonstrate that our adaptive operator improves the coverage while maintaining high precision and competitive cost efficiency.

2 RELATED WORK

2.1 Similarity-based Joins

Similarity-based joins identify candidate value pairs whose textual similarity exceeds a predefined threshold. Many researchers has explored related algorithms for edit-distance and token-based similarity, especially following the classical paradigm. Early methods include prefix-filtering[1], which support scalable edit-distance joins as embodied in systems such as VChunkJoin [18] and PASS-JOIN [9]. Subsequent efforts introduced partition-based and trie-based indexing structures, such as Trie-Join [16], to further reduce candidate generation costs over large collections of strings. Parallel frameworks such as MassJoin and Fuzzy Join [2] extend these ideas to distributed environments, enabling approximate joins over massive datasets.

While these approaches perform well on purely syntactic variations, they remain limited to surface-form similarity. In addition, their reliance on a global similarity threshold makes them brittle under domain shift and heterogeneous naming conventions, where the appropriate threshold can vary widely across datasets. As a result, similarity-based joins are inadequate for capturing the broader semantic equivalences that arise in real-world tabular data.

2.2 Transformation-based Approaches

Transformation methods aim to synthesize programs that map source values to a target format based on a small set of examples. Classical research such as FlashFill[6] and BlinkFill[15] generate substring-based transformation programs from user-provided examples, offering fast and interpretable transformations but relying heavily on clean, accurate examples. Subsequent work focuses on improving robustness to noise and reducing the large search space inherent in program synthesis. Auto-join[21] introduces predefined transformation units and partitions noisy examples into subsets to constrain the search, while CST [12] further restricts candidate programs by extracting common textual patterns between source and target examples. More recent efforts explore reinforcement learning for transformation discovery. QJoin [17] learns transformation strategies across join tasks by training an RL agent with a uniqueness-aware reward and reusing successful transformation chains through agent transfer and transformation caching. While improves the efficiency and reuse of transformation operators, it remains limited to textual transformations and does not address semantic equivalences that cannot be expressed through string-level operations. These methods represent the dominant trend of limiting the transformation space to achieve better runtime and noise handling. However, because they operate exclusively on string-level primitives, their ability to generalize is fundamentally constrained, and they often miss valid transformations that require semantic reasoning or knowledge beyond surface-form matching.

2.3 LLM-Assisted Methods

Recent advances in large language models (LLMs) have motivated their use in tabular transformation and semantic normalization, as they capture contextual and semantic relationships beyond what fixed transformation rules or similarity metrics can achieve. Early LLM-assisted methods use example-driven pattern induction to produce standardized values, while more advanced frameworks such as DTT [5] directly predict the target-format value for each input. By training on large collections of synthetic transformation patterns, DTT generalizes across diverse formatting rules and avoids the heavy program search required by classical approaches. Nonetheless, LLM-based methods remain computationally expensive, often assume consistent input patterns, and struggle when column values are highly heterogeneous. They also apply the same reasoning strategy to all rows and lack mechanisms to identify when simpler syntactic or embedding-based operations would suffice. Thus, while LLMs expand the range of transformations that can be modeled, they still do not provide an adaptive way to choose the most appropriate strategy for different value types.

3 DEFINITION OF THE PROBLEM

We aim to identify matches between two columns whose values represent the same real-world entities but appear in different syntactic or semantic forms. Let

$$A = \{a_1, a_2, \dots\}, \quad B = \{b_1, b_2, \dots\}$$

denote the sets of unique values in the two columns to be joined. Unlike equi-joins, which require exact string equality, our goal is to find all value pairs that refer to the same entity even when their surface forms differ (e.g., "USA"–"United States of America", "The Netherlands"–"Netherlands").

Formally, we seek to produce a set of match pairs

$$R = \{(a_i, b_j) \mid a_i \in A,\ b_j \in B,\ a_i \equiv b_j\},$$

where $\equiv$ denotes semantic equivalence independent of syntactic similarity. Such equivalence may arise from abbreviations, aliases, typos, rephrasings, or domain-specific knowledge (e.g., "South Korea" $\leftrightarrow$ "Republic of Korea").

The challenge is that existing techniques capture only subsets of these discrepancies: rule-based transformations handle limited syntactic variations; similarity-join methods depend on unstable thresholds; and LLM-based normalization can be costly in terms of both runtime and monetary expense when applied to all candidate pairs.

Therefore, the problem addressed in this work is to develop a *generic, accurate, and scalable module* that maximizes valid value matches across heterogeneous discrepancy types while remaining efficient.

4 OUR METHOD

Our approach aims to maximizes the number of valid value matches between two columns by dynamically selecting the most appropriate matching strategy for each case. Rather than relying on a single

similarity measure or transformation rule, the proposed operator will integrate multiple complementary value-matching methods and a decision module that determines which one to apply based on the characteristics of the value pairs.

The core of the system is a selector module that dynamically decides which matching primitive to apply. This module is designed as an independent component, capable of integrating various decision mechanisms. In this project, we focused on implementing a RL-based selector, where the agent learns to choose the optimal primitive by interacting with the environment and receiving feedback based on the quality of the produced matches. The reward signal is determined by whether the prediction is correct, the number of attempts required to reach the solution, and the cost associated with each primitive.

Inspired by the strong performance reported in other works [20] [8], we use a RL approach for adaptive operator selection. Below, we describe how this value matching problem can be formulated as RL task.

State Representation. The state s encodes metadata and statistical descriptors of the source value and the candidate target values. These features serve as hints to guide model selection: (1) number of target values, captures the complexity in terms of cardinality; (2) edit-distance statistics, reflects the variability of the terms; (3) value type, semantic category of the values (e.g., organization, place), providing contextual information about the source and target; (4) history of applied algorithms, records which primitives have been used, helping to avoid unnecessary repetition.

Action Space. The action a corresponds to selecting one of the seven value matching primitives used to compare the source and target values. Each action triggers a different matching strategy with distinct computational costs and capabilities. Formally,: $a \in \{lexical, semantic, llm, shingles, regex, identity, accent\_fold\}$. These primitives provide complementary ways to evaluate similarity, enabling the agent to choose the most suitable method for each matching attempt.

Reward Function. After applying the chosen primitive, the system compares the predicted matches to the ground truth and computes a reward:

$$r = \begin{cases} 1.0 - 0.3 \times (1 - attempts) - \text{cost} & \text{if correct prediction} \\ -1.0 & \text{if max attempts reached} \\ -0.1 & \text{if incorrect prediction} \end{cases}$$

where $attempts$ is the number of attempts made to find a valid match, and $cost$ is the cost of the selected primitive.

Policy Learning. The agent aims to learn a policy $\pi(s) \rightarrow a$ that maximizes the expected cumulative reward. We employ the PPO implementation from RLlib[11]. Training proceeds as follows: a feature-based state representation is first extracted for each source–target pair; the agent then selects an action, corresponding to a value-matching primitive, according to the current policy; the selected primitive is executed to generate a predicted match; the reward is computed using the ground-truth labels; and finally, the policy network is updated based on the observed reward. Over time, the agent learns which primitives are most effective for specific column characteristics, enabling adaptive and data-aware selection of matching strategies.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate our operator through a series of experiments designed to test its accuracy, robustness, and ability to generalize across different types of value discrepancies. Our goal is to determine whether the adaptive operator improves semantic matching performance, how it compares with similarity-based and transformation-based baselines, and whether the RL-based selector works reliably across diverse domains.

5.1 Datasets

To evaluate the effectiveness of our method, we use four pre-existing real-world datasets, for a total of 271 tables.

Auto-FuzzyJoin DBPedia Datasets (AUTOFJ). We employ the fuzzy-join benchmark introduced by Li et al. [10]. This consists of 50 datasets constructed from annual snapshots of DBPedia entities between 2013 and 2016. A DBPedia entity has a static `"entity-id"`, an `"entity-name"` (a title of a Wikipedia article), and an `"entity-type"` that describes the entity's broader category. For example, an entity name could be `"Kosovo"`, and the entity type would be `"Country"`.

The open-source nature of Wikipedia results in the evolution of entity names over time; for example, the Kosovo article was renamed to `"Kosovo (region)"`, producing a real-world fuzzy-match scenario. As such, the authors used differences in entity names across snapshots to create fuzzy-match scenarios. For each entity type, a table is generated for each year, and cross-year table pairs form fuzzy-join tasks with ground-truth matches determined by the invariant entity identifiers. The benchmark uses the 2013 snapshot as the left relation and the union of subsequent snapshots as the right relation, introducing challenging cases such as many-to-one matches and missing counterparts [10]. We use all 50 datasets as provided to ensure comparability with prior work.

Web Tables Dataset (WT). We also evaluate on the Web benchmark introduced by Zhu et al. [21]. This collection consists of table pairs extracted from Web data. The authors sampled table-intent search engine queries, such as "`list of US presidents`", and used Google Tables to retrieve Web tables associated with each query. For each topic, they selected table pairs that reference the same entities but use different textual or structural conventions, such as `"White Christmas"` and `"White Christmas - Bing Crosby"` referring to the same song. Because Web tables often contain inconsistent formatting, such as name variants or differing attribute conventions, these datasets present particularly challenging join scenarios [21]. Its challenging nature has resulted in its use as a benchmark in several papers since its introduction [5]. The benchmark spans 17 topics and includes 32 pairs of tables; we use the table pairs exactly as released.

Spreadsheets Dataset (SS). Adapted by Dargahi et al. from the 2016 Syntax-Guided Synthesis Competition [5], the Spreadsheets dataset contains 108 tables collected from the Microsoft Excel product team and user help forums, reflecting real data cleaning tasks encountered by Excel users. Each table consists of columns that represent the same information, but in different formats. For example,

a "firstnames" table consists of users' full names mapped to their first names. The table values are noticeably short, at an average of 19 characters per input field. Compared to Web-sourced data, these spreadsheets exhibit less noise and fewer textual inconsistencies, making for a less challenging benchmark [5].

Knowledge Base Web Tables (KBWT). Also adapted by Dargahi et al. [5], the KBWT collection is comprised of tables extracted from a structured knowledge base. Values typically requires semantic transformations or background knowledge to be correctly aligned. For example, an "AwardToArtist" table maps recording artists to each album certification they have, with "2x-Platin" mapping to "Abba", "AC/DC", "Guns N' Roses", and 11 other artists. Ground truth correspondences are manually annotated by the original authors.

As opposed to textual variation, the Knowledge Base Web Tables dataset emphasizes deeper semantic relationships. Following prior work, we use the single-column tasks from this collection, totaling 81 tables. On average, tables are long but rows are short, with an average of 113 rows per table and 13 characters per input value [5]. This dataset enables us to examine performance on cases that require semantically informed transformations beyond text-based normalization.

5.2 Metrics

We evaluate the quality of the predicted matches using four standard metrics: accuracy, precision, recall, and F1-score. For each source value, the model outputs a single predicted target value, which is compared against the gold label.

Accuracy measures how often the model predicts the correct target value:

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{source values}}.$$

For pair-level evaluation, we denote by TP the number of correctly predicted matches (true positives), by FP the number of incorrect predicted matches (false positives), and by FN the number of missed gold matches (false negatives). Precision, recall, and F1-score are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

All metrics are computed at the value level and reported for our method and the baselines.

5.3 Comparison of Value Matching Methods

We train our model using 271 tables drawn from the four benchmark collections described earlier. For each table, we construct a set of "source-target" value mappings. We use 80% of these mappings for training and the remaining 20% for validation. During evaluation, the model predicts a normalized value for each source entry, and a prediction is considered correct if it matches the gold target value associated with that entry.

We evaluate the performance of our model on the basis of accuracy, precision, recall, and F1-Score. We consider a prediction correct if it matches the gold value associated with the source value.

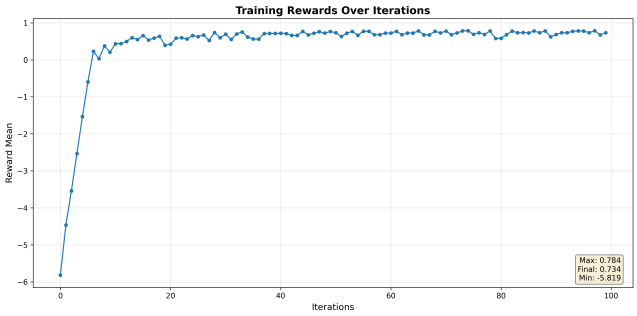

Figure 1: Mean reward obtained by the agent over training iterations.

In Figure 1 shows the evolution of the mean reward over episodes in each iteration.

Table 1 compares our method with the Auto-FuzzyJoin [10] and DTT [5] baselines across four datasets. Beyond achieving the highest or near-highest scores on all metrics, our method shows a clear advantage in handling challenging, noisy matching scenarios. On AUTOFJ and KBWT, where surface similarity is unreliable, both traditional heuristics and embedding-based baselines degrade sharply, while our method maintains strong F1 performance, indicating better robustness to irregular or sparse patterns. On the cleaner SS and WT datasets, our method continues to match or exceed the baselines, achieving perfect precision and stable recall. These results suggest that our approach captures more resilient matching cues that generalize across both structured and highly noisy datasets, leading to the strongest overall average performance.

Table 2 shows the results for each primitive and our method on the four datasets. Across all datasets—and confirmed in the aggregate results—the LLM-based method achieves the highest accuracy, precision, recall, and F1-score, consistently outperforming the lexical, semantic, regex, and shingling baselines.

However, our model does not simply choose the most accurate primitive. Instead, it explicitly incorporates the computational cost of each method when selecting an action. The RL policy is designed to prefer cheaper primitives (e.g., `identity`, `lexical`, or `accent_fold`) whenever they are sufficient to correctly solve a given instance, and only escalate to more expensive methods (such as the LLM primitive) when necessary.

Our objective is therefore not to beat the LLM primitive outright—it is already the strongest individual method—but rather to learn a cost-aware policy that balances effectiveness with computational efficiency, choosing the cheapest correct primitive for each value-matching task.

The example below illustrates how our model processes an input value (Source), compares its prediction (Predicted) to the ground-truth target (Gold), selects a primitive (Action) to perform the match, and receives a corresponding Reward.

As shown, for more challenging cases—such as matching `+25,9 98-898-180` to `25`—the policy selects the `llm` primitive. Although this method is more computationally expensive, the model learns that simpler primitives are unlikely to succeed for such heterogeneous inputs.

Dataset	Our Method				AutoFuzzy Baseline				DTT Baseline			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
AUTOFJ	0.814	1.000	0.814	0.900	0.600	0.896	0.642	0.731	0.514	0.518	0.509	0.514
SS	0.933	1.000	0.933	0.966	0.836	0.916	0.864	0.881	0.938	0.938	0.938	0.938
WT	0.937	1.000	0.937	0.968	0.887	0.943	0.940	0.931	0.915	0.916	0.915	0.915
KBWT	0.367	1.000	0.367	0.537	0.160	0.307	0.179	0.205	0.115	0.139	0.109	0.115
Avg	**0.763**	**1.000**	**0.763**	**0.843**	0.597	0.766	0.650	0.687	0.621	0.628	0.616	0.621

Table 1: Comparison of our method with Auto-FuzzyJoin and DTT baselines across four datasets.

Method	AUTOFJ				SS				WT				KBWT				Aggregate			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
lexical	0.612	0.784	0.762	0.759	0.665	0.665	1.000	0.798	0.688	0.688	1.000	0.815	0.066	0.066	1.000	0.123	0.508	0.508	1.000	0.624
semantic	0.739	0.863	0.855	0.850	0.712	0.712	1.000	0.832	0.890	0.890	1.000	0.942	0.132	0.132	1.000	0.233	0.618	0.618	1.000	0.714
llm	**0.815**	0.851	**0.951**	0.898	**0.933**	0.968	**0.963**	**0.966**	0.888	0.968	0.915	0.941	**0.367**	0.470	**0.626**	**0.537**	0.751	0.814	**0.864**	0.835
shingles	0.698	0.830	0.824	0.822	0.887	0.887	1.000	0.940	**0.945**	0.945	1.000	**0.972**	0.092	0.092	1.000	0.168	0.655	0.655	1.000	0.726
regex	0.670	0.812	0.805	0.802	0.519	1.000	0.519	0.683	0.781	0.962	0.806	0.877	0.042	0.367	0.045	0.081	0.503	0.762	0.569	0.611
identity	0.000	0.000	0.000	0.000	0.032	1.000	0.032	0.062	0.011	1.000	0.011	0.021	0.000	0.111	0.000	0.001	0.011	0.528	0.011	0.021
accent_fold	0.098	0.192	0.180	0.178	0.032	1.000	0.032	0.062	0.011	1.000	0.011	0.021	0.000	0.111	0.000	0.001	0.035	0.764	0.035	0.066
our method	0.814	**1.000**	0.814	**0.900**	**0.933**	**1.000**	0.9334	**0.966**	0.937	**1.000**	**0.937**	0.968	**0.367**	**1.000**	0.367	**0.537**	**0.763**	**1.000**	0.763	**0.843**

Table 2: Comparison of all primitives across the AUTOFJ, SS, WT, and KBWT datasets with the aggregate mean performance

Conversely, for simpler transformations—e.g., `JacqualineOlague` → `Jacqualine` or `+106,769-858-438` → `+106,(769),858-438`—the policy selects lightweight primitives like `shingles` or `lexical`. This demonstrates that the agent learns to adaptively choose the cheapest correct method for each input rather than relying on a single expensive primitive.

Source: `+106 769-858-438`
Gold: `+106 (769) 858-438`
Predicted: `+106 (769) 858-438`
Action: lexical
Reward: 0.93

Source: `Jacqualine Olague`
Gold: `Jacqualine`
Predicted: `Jacqualine`
Action: shingles
Reward: 0.93

Source: `+25 998-898-180`
Gold: 25
Predicted: 25
Action: llm
Reward: 0.70

6 DISCUSSION AND CONCLUSION

We proposed a generic value matching operator that integrates multiple value matching methods and adaptively selects which one to apply. By conditioning the RL agent on column-level metadata and rewarding both accuracy and efficiency, our agent learns to deploy different primitives in contextually appropriate ways. Our experiments show that the agent not only improves matching performance, but also uses expensive methods prudently. For example, on the Spreadsheets Dataset—which has relatively low semantic complexity and minimal reliance on external context—the agent correctly favors the shingles primitive over the LLM primitive. This behavior demonstrates that the RL policy internalizes trade-offs between accuracy gains and computation cost, rather than mechanically applying the most powerful method.

Despite these promising results, some limitations remain. The universe of primitives is limited and cannot capture all transformation patterns encountered in real data. Certain mappings, such as `"Edward Davis"` to `"e.davis"`, invoke the LLM primitive, even though they could be better handled by a fine-tuned language model or more robust transformation algorithm. Additionally, although computational costs for each primitive method can be expressed in terms of runtime, the corresponding reward coefficients require manual tuning. Finding the appropriate balance between accuracy and cost proves difficult, and suboptimal tuning can bias the agent towards overly cheap or overly expensive primitives. Finally, although our agent learns effective policies for the many domains represented in our training data, shifting to a new data domain could reduce performance. However, the availability of synthetic data generation tools could be used to generate policies that generalize more robustly.

Our results suggest that reinforcement learning offers a promising alternative to transformation-heavy methods and LLM-centric approaches. By explicitly optimizing for both accuracy and cost, an RL agent can act as a flexible controller that integrates multiple primitive methods and adapts to the structure of the data at hand. We view this as a step toward efficient, domain-aware, and interpretable value-matching systems.

7 CONTRIBUTIONS OF EACH MEMBER

- Roque Lopez: Contributed to the design and development of the RL approach, executed the training and evaluation jobs, implemented some primitives, analyzed the feature set, and contributed to multiple sections of the paper.
- Thiago Viegas: Worked on primitive methods used by the agent to identify the best, implemented feature extraction methods, metrics generation for our solution, contributed to literature and final presentation.

- Amory Gao: Loaded data, processed and standardized values, identified source and target columns, split into training and testing subsets, and contributed to paper and presentation.
- Remo Shen: Executed baseline experiments, conducted literature review, and contributed to writing the introduction, related work, and other documentation.

REFERENCES

[1] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A primitive operator for similarity joins in data cleaning. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 5–5.

[2] Zhimin Chen, Yue Wang, Vivek Narasayya, and Surajit Chaudhuri. 2019. Customizable and scalable fuzzy join for big data. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2106–2117.

[3] Graham Cormode and Shan Muthukrishnan. 2007. The string edit distance matching problem with moves. *ACM Transactions on Algorithms (TALG)* 3, 1 (2007), 1–19.

[4] Arash Dargahi Nobari. 2025. Transforming Tables for Heterogeneous Joins: From Common Strings to Complex Mappings. (2025).

[5] Arash Dargahi Nobari and Davood Rafiei. 2024. DTT: An Example-Driven Tabular Transformer for Joinability by Leveraging Large Language Models. In *Proceedings of the ACM on Management of Data*. 1–24.

[6] Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745* (2022).

[7] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *Nature* 645, 8081 (2025), 633–638.

[9] Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. 2011. Pass-join: A partition-based method for similarity joins. *arXiv preprint arXiv:1111.7171* (2011).

[10] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data*. 1064–1076.

[11] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[12] Arash Dargahi Nobari and Davood Rafiei. 2022. Efficiently transforming tables for joinability. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1649–1661.

[13] Atish Pawar and Vijay Mago. 2018. Calculating the similarity between words and sentences using a lexical database and corpus statistics. *arXiv preprint arXiv:1802.05667* (2018).

[14] Magnus Guldberg Pedersen, Benjamin Kock Fazal, and Kyoung-Sook Kim. 2024. Optimizing Semantic Joinability in Heterogeneous Data: A Triplet-Based Approach with Pre-trained Deep Learning Models. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 6092–6100.

[15] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment* 9, 10 (2016), 816–827.

[16] Jiannan Wang, Jianhua Feng, and Guoliang Li. 2010. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1219–1230.

[17] Ning Wang and Sainyam Galhotra. 2025. QJoin: Transformation-aware Joinable Data Discovery Using Reinforcement Learning. *arXiv preprint arXiv:2512.02444* (2025).

[18] Wei Wang, Jianbin Qin, Chuan Xiao, Xuemin Lin, and Heng Tao Shen. 2012. Vchunkjoin: An efficient algorithm for edit similarity joins. *IEEE Transactions on Knowledge and Data Engineering* 25, 8 (2012), 1916–1929.

[19] Peter Yeh, Bruce Porter, and Ken Barker. 2004. Mining transformation rules for semantic matching. In *ECML/PKDD 2nd International Workshop on Mining Graphs, Trees, and Sequences*, Vol. 10. 945645–945671.

[20] Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025. DeepAnalyze: Agentic Large Language Models for Autonomous Data Science. *arXiv preprint arXiv:2510.16872* (2025).

[21] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.