COVER LETTER OF PAPER 71

Dear Meta Reviewer and Referees:

We have substantially revised the paper following the Referees' suggestions. For the convenience of Referees, changes for addressing the concerns of R1, R2 and R4 are color-coded in blue, orange and purple, respectively. We would like to thank the Referees for insightful comments and for supporting this work!

Below please find our responses to the reviews.

Response to the comments of the Meta Reviewer.

[Meta summary & recommendation]

- The description of the approach is difficult to understand and could be clarified using concrete examples.

- The experimental evaluation fails to include models that are more resilient to outliers and did not make use of split validation; for the results to be credible, it is crucial to keep a hold-out test split.

- The authors should conduct additional experiments on the explainability of OMRs in Section 6.

C1: for the experiments, you must use a new validation-split

C2: conduct additional experiments on the explainability of OMRs (as per discussion)

[A] Many thanks! As suggested, we have revised/added examples to clarify our approach (pp. 1,3-7), repeated the experiments under the train-validation-test setting (pp. 8-13), tested new baselines TabPFN and XGBoost (resilient to outliers), and added additional experiments on the explainability of OMRs to Section 6 (pp. 12).

Availability of artifacts

[R2 & R4] *Please provide an anonymous Git repository.*

[A] Thanks! The anonymous code access link for this paper is https://anonymous.4open.science/r/code-3B65/.

Response to the comments of Referee #1.

[R1O1] *It is unclear to me why the proposed technique is specific to relational data. Can't it work for non-relational data? e.g., just a flat table, which is the most common format for ML?*

[A] Thanks! We have clarified that our method is not limited to relational databases (pp. 2). The method can be applied to any tabular data format, stored in databases or simple files like TSV, CSV or Excel. It aims to support various ML applications, where train/test/validation data is often organized in flat, tabular formats.

[R1O2] *Example 1 is very confusing. Grade is a direct function of GPA, according to the description. Then what's the point of using other attributes to predict Grade? I recommend the authors come up with a better example where the target attribute indeed requires all attributes for prediction and not just one attribute. Also, I don't understand the rationale behind "Thus, if the Absence of t1 is correct, then its StudyTimeWeekly is an outlier as it is inconsistent with other features". Couldn't the Absence be an outlier instead of StudyTime? How are you deciding which attribute is an outlier here? For t4, since their GPA is 4.00, they should be marked as A, which happened. I don't follow why "since its label A is rare among all tuples". The claims in Example 1 is very arbitrary and fails to motivate the work. Overall, Example 1 does a really poor job communicating what's an Ugly outlier. Better example is needed.*

[A] Thanks! We have used a real-life example for diabetes diagnosis as Example 1 (pp. 1). Specifically, an ML classifier diagnoses diabetes based on a relation, where different types of outliers affect the prediction of a trained classifier. We show cases where misclassifications occur due to outliers in critical features, incorrect labels, and rare data patterns, which demonstrate the concept of "ugly outliers" in the context of diabetes diagnosis. The revised example also illustrates the importance of considering multiple data points for prediction (pp. 3) and the challenges in identifying outliers.

[R1O3] *I don't follow "not only attributes can be "ugly" outliers, but also classification labels." How can classification labels be outliers, they are the misclassifications, and the goal is to improve classification my fixing outliers, but not the classification directly, at least that was my understanding. If you are talking about repairing classification outputs, then this becomes a different problem.*

[A] Thanks! We have clarified that label outliers are in training data (pp. 1 and 3). The classification labels in supervised training data can be outliers, as there are inevitably mislabeled samples due to knowledge limitation or carelessness of human annotators [124]. This has been a longstanding challenge in the ML community, where "classification with noisy labels" remains a research direction with theoretical and practical significance in the last decade [65, 114]. Different from ML researchers who mainly focus on designing noise-tolerant models, we aim to detect and fix ugly outliers in the training data such that the overall performance of downstream ML models can be improved. We would like to clarify that we do not repair the classification outputs of ML models, where our OMRs only manage and improve the training data for ML models.

[R1O4] *How different is OMR's logical predicate part from traditional denial or integrity constraints? If they are the same, the paper should acknowledge re-use of existing work for that part. Also, technique wise, can traditional denial constraint discovery approaches be used to discover OMRs?*

[A] Thanks! We have clarified this (Section 2, pp. 4; Section 4, pp. 4).

- The logical predicates of OMR use standard comparison operators like denial constraints (DCs). Unlike DCs, they reduce false positives and negatives of ML predictions when put together with statistical functions and ML predicates.
- As remarked in Section 4, discovery methods for DCs are not applicable to OMRs, since they do not support mining function predicates, ML predicates and their hyperparameters in OMRs. Moreover, OMRs support multi-variable rules across multiple tables, which current DC discovery algorithms do not support.

[R1O5] *If ohunt is "referencing external data sources", then what are the assumptions about availability of such external data sources? The setup and assumptions of the work needs to be clarified upfront.*

[A] Thanks! We have clarified in Section 5 (pp. 6) that referencing external sources KB is optional. When KB is available, we align entities shared by $\mathcal{D}$ and KB via standardized access interfaces of KB (*e.g.*, Wikidata's SPARQL endpoint, Kaggle's API) and mitigate (small number of) noises in KB via the error-correction of OFix. If

KB is unavailable, our method falls back on data-driven correction.

[R1O6 & O8] *"a method to fix outliers with right values, instead of simply removing tuples" -> How is it different from existing techniques for missing value imputation?*

The novelty about the Outlier Fixing part (Section 5) is unclear to me. Why can't one use standard ML prediction models to predict the value to replace the detected outlier value? How is it different from standard missing value imputation? The challenges in fixing outliers using existing techniques aren't sufficiently discussed and needs improvement. Also, how is it different from standard data cleaning under some constraints? Such as HoloClean?

[A] Thanks! We have clarified the novelty of OFix in Section 5 (pp. 8) as follows. Unlike existing data imputation methods,

(1) we advocate for a combination of logical reasoning and ML prediction. While ML-based missing-value imputation (Section 5.1) is one of our strategies, logic predicates can reduce its false positives and false negatives, as verified by prior work [50] and our own ablation study (0.5% more accurate than ML alone in Section 6).

(2) We extend the chase to recursively propagate fixes, allowing earlier corrections to refine subsequent outlier adjustments. The process is "deterministic" (Church-Rosser [6]), *i.e.,* the process always converges to a unique result, regardless of execution order. It also guarantees correctness under certain conditions (pp. 7).

(3) We fix only ugly outliers, rather than all missing values. These distinguish our method from heuristics like HoloClean [142].

(4) Logic and function predicates enhance the interpretability of outliers and ML predictions, as illustrated by examples in Section 2.2.

[R1O7] *There are works that also blames a certain attribute value for model prediction and similar results, for example "Fariha et al. ExTuNe: Explaining Tuple Non-conformance", and other works that explains outliers such as "Miao et al. CAPE: explaining outliers by counterbalancing". Generally, ML explainability approaches that blame certain data tuples/attribute values for an observed results are related to this work and needs more discussion.*

[A] Thanks! We have cited the suggested methods, including ExTuNe and CAPE in the "Explainable AI" part of related work (Section 7, pp. 13), and compared them experimentally with our approach (Section 6, pp. 12). Specifically, CAPE is limited to detecting outliers and cannot fully explain different types of outliers. ExTuNe identifies key features for explaining outliers, but it does not distinguish between different types. Neither baseline can differentiate good outliers from combination-type ugly ones. In contrast, OMRs identifies, distinguishes, and explains various types of outliers.

[R1O9] *The tested classifiers are very old-school such as Decision Tree, SVM, etc. It is not justified why more modern approaches such as deep learning wasn't used.*

[A] As suggested, we have added comparison with deep learning (DL) model Tabpfn [83] (Section 6, pp. 8). We should have clarified that we did not use DL baselines since previous study [83] shows that DL methods are not suitable for tabular data classification for the following two reasons: (1) small datasets are common in real-world applications, where most DL methods are limited when given inadequate data [B1], and (2) feature correlation in tabular data varies between datasets and ranges from independent to highly correlated, which threat the performance of DL methods [B2].

[B1] Léo Grinsztajn, et al. "Why do tree-based models still outperform deep learning on typical tabular data?" NeurIPS 2022.
[B2] Vadim Borisov, et al. "Deep neural networks and tabular data: A survey". IEEE Trans. on neural networks & learning systems 2022.

[R1O10&O13] *What values are being reported in Tables 3 and 4 aren't clear from the caption. Is it model accuracy or fraction of detected outliers?*

Please make the captions of all figures and tables more self-contained. For example, what are m and n in Figure 2 (o) and (p)?

[A] As suggested, we have extended captions of all figures and tables, including those for Tables 3 and 4 (F1-score for ugly outlier detection, pp. 8) and m and n in Figure 2 (o) and (p) (m for the attribute sampling ratio and n for the tuple sampling ratio, pp. 10).

[R1O11] *It will be good to highlight the key takeaways in the experimental section.*

[A] Thanks! We have substantially expanded the summary of experiments to highlight the following key points (pp. 12-13): (1) the distribution of good, bad and ugly outliers in real-world data; (2) the rarity of ugly outliers and their impact on ML classification; (3) OHunt's effectiveness in detecting rare ugly outliers; (4) OHunt's robustness to diverse noise types: local, global, dependency, and clustere; (5) the benefit of combining logical reasoning with ML/function predicates to boost classification accuracy; (6) the factors contributing to OHunt's effectiveness, robustness, and scalability in outlier detection, correction, and rule learning; (7) the reason why uOMRs are more robust and efficient than sOMRs.

[R1O12] *Figure 2 (m) and (n), the top ones: the patterns are very similar for ICL and UOMR.*

[A] Thanks! We have replaced the legends for ICL and uOMRs in Figures 2(m) and 2(n) with more distinct legends (pp. 10).

[R1O14] *I understand that the impact of one data point is used to detect ugly outliers. But what if a few data points, together, act as ugly outliers? Will the proposed technique generalize to that?*

[A] Thanks! We have clarified in Section 2.2 (pp. 3) that if a combination of data points are "ugly" when taken together, but each of the points is not ugly on its own, OMRs can (a) express the combination as precondition and (b) infer ugly outliers in their consequence. In particular, a real-life example there showcases how OMRs catch a combination of data points that make an ugly outlier.

Many thanks for your support and insightful suggestions!

Response to the comments of Referee #2.

[R2O1] *The experimental study of the paper can be improved in a few ways.*

Base Classifiers. While the list of classifiers studied (SVM, RF, etc.) are a good start, these are not necessarily state-of-the-art classifiers today. For example, boosting tree methods such as XGBoost (which are often less sensitive to outliers) or pre-trained tabular networks *like TabPFN, are some of the known methods that perform well on prediction tasks for tabular data. Adding these two as base classifiers (M) in the experiments would better show the practical benefit of the proposed method, in the presence of state-of-the-art ML classifiers.*

Results on all datasets. The authors mention that 18 real datasets are used in the experiment, only 4-5 are shown in Figure 2 and Table 5. While I understand that it is hard to show all results with limited real estate, it may be better to at least report an 18-dataset-average in these experiments (in addition to results on 4-5 individual datasets), just so that readers have a comprehensive picture of the performance of different algorithms on a large set of benchmarks.

Split validation. The authors mention "We used multi-class datasets (e.g., DryBean) and real outlier datasets (e.g., Shuttle), splitting them into 70% training and 30% testing sets." If there is only train/test, and test is exposed to the proposed algorithm to "search" and find the best cleaning strategies (which seems to be the case based on descriptions in Section 3), it is not entirely surprising that quality results on the test is improving. It would be better to avoid exposing test, and use train/validation/test splits instead.

There are experiments on whether outliers are correctly classified as bad/ugly, etc., It is not obvious how these bad/ugly labels are obtained, as there may not be ground-truth labels, and the exact labeling can be subjective. Additional details on how the labels are obtained would seem necessary.

The DC charts in Figure 1 do not seem to show the difference between the proposed method and any other methods as statistically significant. It is unclear why this is the case.

In Figure 2, sOMRs are showing as not competitive. Some discussions in this area would be helpful

Missing related work. For example, the database literature, CleanML [1] and its follow-ups study the same problem of cleaning data for ML, which is not discussed or experimentally compared with. [1] CleanML: A study for evaluating the impact of data cleaning on ml classification tasks, in ICDE 2021.

[A] Thanks! We have revised the experimental part as follows.

(1) We have added experiments on more advanced baseline classifiers, *i.e.,* XGBoost and TabPFN, in Section 6 (pp. 8-13).

(2) We have reported the 18-dataset-average in Section 6 (pp. 8-13), plus a couple of individual cases. Many thanks for the suggestion!

(3) Thanks! As suggested, we have randomly split the dataset into 70% for training, 20% for validation, and 10% for testing. The training set is used to train the models and learn OMR rules from the data. The validation set helps select hyperparameters in Bayesian optimization and choose cleaning strategies, but does not directly participate in model training. The test set is used solely for the final evaluation of model performance. We have reported the performance for all methods on the test set, including metrics such as accuracy, robustness, and scalability in Section 6 (pp. 8-13).

(4) We have provided the following details in Section 6 (pp. 8). (a) The ground truth for outliers is available in all the datasets tested, which had been annotated by domain experts and widely used in ML studies [B3-B5]. (b) We label ugly outliers through experimental validation, such as those that cause misclassifications, or violate either domain knowledge or business rules. (c) We label good outliers if they are legitimate (*i.e.,* they comply with domain knowledge) but occur very infrequently, and if they help the model learn long-tail distributions. Removing them would impair the model's ability to recognize rare patterns (as validated by our ablation test, pp. 11), and they are consistent with the long-tail distribution. (d) The remaining outlines are all labeled as bad ones.

[B3] Han, Songqiao, et al. "Adbench: Anomaly detection benchmark". NeurIPS 2022.

[B4] Domingues, Rémi, et al. "A comparative evaluation of outlier detection algorithms: Experiments and analyses". Pattern recognition 2018.

[B5] Li, Zheng, et al. "Ecod: Unsupervised outlier detection using empirical cumulative distribution functions". TKDE 2022.

(5) Thanks! We have found that there was missing plot data represented by NaN in the plotting code. Adding the missing data, we have re-plotted Figure 1 (pp. 9) with correct data and codes, which shows significant statistical differences between the algorithms.

(6) We have clarified this in Section 6 (pp. 13). As shown there, sOMRs significantly outperform other label-based methods. However, they show no consistent advantage over unsupervised uOMRs on some datasets due to: (a) limited outlier labels in small datasets, which can lead to overfitting and reduced performance for sOMRs on smaller datasets, (b) potentially incorrect labels, as commonly found in practice; and (c) uOMRs benefiting from signals generated during the chase process by OLeaner and OFix, enabling them to detect and repair outliers effectively without labeled guidance. As a result, uOMRs are often faster and more robust.

(7) Thanks! We have expanded our discussion on data cleaning for ML (CleanML, pp. 13), and experimentally compared CleanML with ours (Section 6, pp. 10-11). As shown in Figure 2 (k-n) (pp. 10), OMRs consistently lead to greater improvements in ML classification accuracy than CleanML, even when both use CART-based detection. This is because OFix provides multiple targeted strategies to repair ugly outliers with accuracy guarantees, whereas CleanML relies on just three basic imputation methods for all outliers.

[R2O2] *The proposed method is complex, and the presentation of the method in Section 3-5 is a bit dry. It would benefit readers if some examples could be used to better illustrate the proposed method.*

[A] Thanks! As suggested, we have provided more examples and case studies in Sections 2-5 to better illustrate our method. Newly added examples include additional OMRs on pp. 3, Example 2 on pp. 5-6, Example 3 on pp. 7, and the case study on pp. 12.

[R2O3] *It is unclear why logic rules are necessary in the proposed approach, when the use case is ML applications (where the prediction quality is often the ultimate metric). Most ML models are not human explainable anyways, so it is unclear whether embed logic rules to make the process explainable is needed. Some ablation studies without logic rules may shed additional light on the importance of this design.*

[A] Thanks! We have clarified the following (pp. 6-8; please also see our response to **R1[O6&O8]**). (1) OMRs combine statistical functions, ML predictions and logical reasoning in a unified framework, to improve the overall accuracy. While ML prediction (pp. 6) is one of our strategies, logic predicates reduce its false positives and negatives, as verified by prior work [51] and our ablation study (pp. 11; 0.5% more accurate than ML prediction alone). (2) In the unified framework, we extend the chase with OMRs to recursively propagate fixes, allowing earlier corrections to refine subsequent outlier adjustments. The process is "deterministic" (Church-Rosser [6]), *i.e.,* the process always converges to unique result, regardless of execution order. It also guarantees the correctness under certain conditions. In contrast, ML prediction alone is often non-deterministic, non-terminating and numerical. We have extended the ablation study (pp. 11) to highlight the need for logical reasoning.

Moreover, logic and function predicates enhance the interpretability of outliers and ML predictions, as illustrated by examples in Section 2.2 (pp. 4) and a newly added real-life case study in Section 6 (pp. 12). Please also see our response to **[R4O2]**.

Many thanks for your support and constructive suggestions!

Response to the comments of Referee #4.

[R4O1] *The section on OMRs is difficult to understand. Consider adding example rules to improve readability.*

[A] Thanks! We have moved example rules from Section 2.3 right after the definition of OMRs (pp. 3-4), to improve the readability.

[R4O2] *To address ugly outliers, the authors focus on the well-studied chase algorithm. However, it is unclear why solving this as an optimization problem was not considered. While ILP may not be relevant due to the non-linear nature of the rules, other optimization methods, like gradient descent, could be applicable or even faster. The rationale for extending the chase algorithm should be explained more clearly.*

[A] Thanks! We have clarified the following in Section 5.2 (pp. 7-8).

(1) We extend the chase, a well-established technique widely used in database applications, to handle outliers for the following reasons (pp. 7-8). (a) We advocate for a combination of logic reasoning and ML prediction with OMRs. This improves ML prediction accuracy by reducing both false positives and false negatives with logic constraints, as verified by prior work [51] and our own ablation study (pp. 11; 0.5% more accurate than ML prediction). Thus chasing the data with OMRs is a natural choice. (b) The chase recursively propagates fixes, allowing earlier corrections to refine subsequent outlier detection and adjustments. It is "deterministic" (Church-Rosser [6]), *i.e.,* despite multiple transformation paths, the process always converges to unique result, regardless of execution order (pp. 7). Please see our response to **R1[O6&O8]** on pp. 8.

(2) Optimization methods in ML also iteratively optimize a target function and approach a solution. Such an iterative process is often (a) non-deterministic: results may depend on initial values, step sizes, and other hyperparameters; (b) non-terminating: it is typically stopped after a fixed number of iterations or when a convergence threshold is met; and (c) numerical: it relies on numerical methods such as gradient descent, rather than logic. In contrast, the chase is deterministic, terminating, and not limited to be numerical. Moreover, it enables us to integrate statistical functions, ML predictions and logic reasoning in a single framework. It is also interpretable, robust, and correct (pp. 8; please see our response to **R4[O5]**).

(3) As shown in Section 5.2 (pp. 7), adding statistical functions and ML predicates introduces new challenges to the classic chase. Thus we extend the chase [6] to ensure termination and determinism.

[R4O3] *The experimental section contains many important takeaways, but it is hard to grasp the overall picture. A brief summary of the results would be extremely helpful. The summary in the end is too short with no explanations for the main findings, mainly highlighting numbers and no take aways.*

[A] Thanks! As suggested, we have further strengthened the summary by highlighting key points (pp. 12-13) such as the distribution of various outliers in real-world data, the rarity of ugly outliers and their impact on ML classification, the benefit of combining logical reasoning with ML/function predicates to boost classification accuracy, and the factors contributing to OHunt 's scalability, robustness effectiveness. Please see our response to **[R1O11]**.

[R4O4] *OMRs can be used to explain ML predictions (Section 2). **Explainable AI** is a well-studied field with multiple established approaches. However, since there are no experiments to support this claim and no specific explanation of the approach is provided, I believe this claim remains unsupported. If the authors wish to retain this point, they should discuss relevant methods in the related work section and back up their claim with experimental evaluation.*

[A] Thanks! We have downplayed the claim (pp. 1) and added (a) discussion about XAI to Section 7 (pp. 13), and (b) new experiments to show how OMRs explain ML predictions to Section 6 (pp. 12).

[R4O5] *The authors should explain what a Church-Rosser means. Additionally, what do you mean by accuracy guarantees? Are there upper or lower bounds on the accuracy of the fixes? The accuracy guarantee part is too high-level, making it unclear exactly what guarantees the algorithm provides.*

[A] As suggested, we have clarified that Church-Rosser property [6]: despite multiple transformation paths, the process always converges to unique result, regardless of execution order (pp. 7).

We have clarified in Section 5.2 (pp. 8) that our accuracy guarantee differs from the numerical (upper or lower) bounds used in statistical ML methods. Since our approach is logic-based, "accuracy" refers to the correctness of the repaired data with respect to the predefined integrity constraints (OMRs). When the chase process terminates, the data is guaranteed to satisfy these constraints, meaning no further violations (errors) exist. The guarantee is binary: the data is either fully consistent with the rules (accurate) or not, rather than measured by numerical error rates or confidence intervals. We have formally proved that our algorithm ensures this logical consistency after repair, providing a strong form of accuracy guarantee within the logical framework of data cleaning.

[R4M1] *Consider adding more examples throughout the paper. While the introduction and experimental section are well-written and easy to follow, other parts of the paper are difficult to fully understand.*

[A] Thanks! We have added more examples and case studies to Sections 2-6 (additional OMRs on pp. 3, Example 2 on pp. 5-6, and Example 3 on pp. 7). Please also see our response to **[R1O2 & R2O2]**.

Many thanks for your support and helpful suggestions!

Outliers: The Good, the Bad and the Ugly

Paper ID: 71

ABSTRACT

This paper studies the impact of outliers in relational dataset $\mathcal{D}$ on the accuracy of ML classifiers $\mathcal{M}$, when $\mathcal{D}$ is used to train and evaluate $\mathcal{M}$. Outliers are data points that statistically deviate from the distribution of the majority. We distinguish good outliers, *i.e.,* novel data, from bad ones, *i.e.,* those introduced by errors. Moreover, we separate ugly ones in influential features from the other bad ones. We find that only the ugly ones have negative impact on $\mathcal{M}$, while the good (resp. the other bad) ones have positive (resp. neglectable) impact. To mitigate the negative impact, we propose a class of rules, denoted by OMRs, to identify ugly outliers by embedding ML outlier detectors and statistical functions as predicates. We develop algorithms to (a) learn OMRs from real-life data, and (b) catch and fix ugly outliers using the learned OMRs, instead of removing tuples. Using real-life data, we empirically show that OMRs improve the accuracy of various classifiers by 7.2% on average, up to 34.8%.

1 INTRODUCTION

Outliers are data points that statistically deviate from the distribution of the majority [24]. Outliers are common [7, 17, 113], *e.g.,* in the ADBench benchmark [73], outliers range from 0.03% [141] to 39.91% [30] in the 57 real-world datasets collected. In light of these, there has been a host of work on outlier detection, based on either ML models [19, 26, 32, 38, 41, 70, 73, 76, 78, 78, 107, 108, 128, 131, 144, 150, 169, 189] or logic rules [11, 18, 37, 47, 139, 173].

This paper studies the impact of outliers on machine learning (ML). Consider a dataset $\mathcal{D}$ for training and evaluating an ML classifier $\mathcal{M}$. We aim to understand what outliers in $\mathcal{D}$ have substantial impacts on the accuracy of $\mathcal{M}$. From our experiments, we find the following. (1) There are "good outliers" that arise from novel data, *i.e.,* data of new distributions that are not included in $\mathcal{D}$. We certainly want to keep such outliers in our training data to make $\mathcal{M}$ capable of recognizing new objects. (2) Outliers may be "bad" as they are introduced by errors. However, we should not delete such outliers in order not to change the data distribution of $\mathcal{D}$, lose important associations between variables [33] and thus reduce the accuracy of $\mathcal{M}$. (3) We need to separate "ugly" ones, *i.e.,* those that result in wrong predictions of model $\mathcal{M}$, from the other "bad" ones. Outliers can distort the decision boundary or change the decision rules of $\mathcal{M}$, especially when the data distribution is uneven or the number of outliers is large, leading to incorrect classifications [7, 88, 92, 190, 191]; *e.g.,* an increase of 4% of outliers in ADBench yields an 11.85% reduction in the accuracy of supervised classifiers [73]. It is the ugly outliers that we need to detect and fix.

Example 1: Table 1 shows a real-world R_{Diabetes}(Age, BMI, Glucose, HbA1c, InsulinResistance, RecordedLabel) relation [97], where RecordedLabel = 1 indicates diabetes. According to ADA guidelines [14], diabetes is diagnosed if at least two of the following hold: (a) Glucose $\geq$ 126 mg/dL, (b) HbA1c $\geq$ 6.5%, and (c) InsulinResistance $\geq$ 5.0. "Lean diabetes" can also occur in patients with low BMI but high insulin resistance [165]. We train a classifier $\mathcal{M}$ (e.g., XGBoost [34]) on Diabetes; its predictions appear in the "PredictedLabel" column of Table 1, alongside the human-provided RecordedLabel. The table highlights several outliers (underlined).

tid	Age	BMI	Glucose (mg/dl)	HbA1c	Insulin Resistance	Recorded Label	Predicted Label
t_1	35	24	145	5.8%	2.0	0	1
t_2	28	22	110	7.0%	6.5	0	1
t_3	37	20	100	6.0%	9.0	1	0
t_4	220	23	105	4.4%	4.0	0	0
t_5	45	27	140	7.1%	4.8	1	1
t_6	35	25	127	6.5%	4.8	1	1
t_7	40	26	125	6.2%	5.1	1	1
t_8	50	27	130	6.3%	4.9	0	1

Table 1: Example Diabetes **relation**

(1) Model $\mathcal{M}$ classifies tuple t_1 as diabetic due to its high Glucose (> 126mg/dL). However, per ADA guidelines [14], Glucose alone is insufficient for diagnosis. This overreliance on Glucose leads to the misclassification of $\mathcal{M}$, making t_1.Glucose an ugly outlier.

(2) Since HbA1c $\geq$ 6.5% and InsulinResistance $\geq$ 5.0, tuple t_2 should be diagnosed as diabetic [14]. However, its recorded label by human is 0, despite accurate feature values. Thus, the recorded label of t_2 is an ugly outlier. Training on such noisy labels harms model $\mathcal{M}$'s performance. Correcting it helps $\mathcal{M}$ learn that high HbA1c and InsulinResistance indicate diabetes, even with normal Glucose. This aligns with research on learning from noisy labels [65, 124], where most ML work focuses on noise-tolerant models [114].

(3) Tuple t_3 has low BMI (20) but high InsulinResistance (9.0), indicating a rare non-obese, insulin-resistant diabetes case. It is a good outlier [165]. Model $\mathcal{M}$ misclassifies it due to limited training examples of this type. Such good outliers help $\mathcal{M}$ generalize to rare conditions and adapt to diverse data distributions [20, 77, 111, 165].

(4) Tuple t_4's Age (220) is a "bad" but not "ugly" outlier, as it does not affect prediction: $\mathcal{M}$ still correctly classifies it as non-diabetic. □

This example raises the following questions: How can we detect different types of outliers? How do we fix outliers that sabotage classification accuracy? Can we repair ugly outliers while preserving data distribution, variable associations, and $\mathcal{M}$'s accuracy?

Contributions & organization. In response to these questions, we propose a framework, named Outlier HUNTer (OHunt), which detects and fixes outliers in relational training data to improve the accuracy of ML classifiers. OHunt has the following features.

(1) Rules (Section 2). OHunt unifies logic reasoning and ML predictions by proposing a class of Outlier Management Rules of the form $X \rightarrow p_0$, denoted by OMRs. As opposed to previous data quality rules [12, 52, 53, 57], the precondition X of an OMR is a combination of (a) logic predicates, (b) ML models for identifying outliers, and (c) statistical and loss functions to provide insights of downstream ML models. Using the functions and ML models, OMRs characterize different types of outliers. Moreover, they are able to reduce false positives (FPs) and false negatives (FNs) of ML detectors for outliers by incorporating logic conditions.

(2) Framework (Section 3). Based on OMRs, OHunt takes as input

an ML classifier $\mathcal{M}$ and a dataset $\mathcal{D}$ for training $\mathcal{M}$. It first learns a set Σ of OMRs from (samples of) $\mathcal{D}$. It then catches and fixes ugly outliers in $\mathcal{D}$ with Σ, with accuracy guarantees (see below). It produces a dataset $\mathcal{D}_c$ in which ugly outliers are fixed, improving the classification accuracy of $\mathcal{M}$ when trained with $\mathcal{D}_c$.

(3) Rule discovery (Section 4). We propose an approach for learning OMRs from real-life data, which selects ML, function and logic predicates in OMRs guided by the classification accuracy of the given model $\mathcal{M}$. We formulate the discovery of OMRs as a meta-learning problem and adopt a Bayesian-based optimization algorithm to identify valuable OMRs. This approach avoids the exponential enumeration of OMR predicates, selects hyperparameters through Bayesian optimization, and employs the accuracy of $\mathcal{M}$ as the criterion to identify effective OMRs for detecting or fixing outliers, instead of traditional metrics such as support and confidence [68, 86, 132].

(4) Detecting and fixing outliers (Section 5). OHunt not only distinguishes ugly outliers from the others, but also fixes the detected outliers. As opposed to simply removing tuples that contain outliers [27, 50, 96, 112, 153], OHunt deduces the right values by a combination of strategies, such as (a) logic reasoning with ground truth, (b) data distribution with ML models, and (c) referencing external data sources. It chases the training data $\mathcal{D}$ with the set Σ of OMRs learned, and accumulates ground truth Γ in the recursive process. It guarantees that the chase converges at the same result no matter what rules in Σ are applied and in what order they apply. Moreover, the fixes generated are logical consequences of the rules in Σ, the ground truth, and predictions of embedded ML predicates. Thus the fixes are correct as long as Σ, Γ and ML predicates are accurate.

(5) Experimental Study (Section 6). Using five classifiers and 38 real-world and synthetic datasets, we empirically find the following. (a) OHunt is accurate in detecting ugly outliers. Its average F1-score is above 0.940, which is 61.9% higher than the baselines, up to 68.5%. (b) It effectively improves the F1-score of classifiers $\mathcal{M}$ by fixing ugly outliers, with an average enhancement of 7.2%, up to 34.8% on the real datasets. (c) It boosts $\mathcal{M}$'s F1-score by 0.5%–4.1% by combining logic and ML/function predicates over using either alone. (d) OHunt is robust to various types of noise. It improves $\mathcal{M}$'s accuracy by an average of 3.8% when injecting different types of noises, up to 13.3% over the baselines on 15 noisy datasets. (e) OHunt is scalable. Its runtime and maximum memory usage grow almost linearly as the parameters of datasets and Bayesian optimization grow.

Novelty. The novelty of OHunt includes (1) a categorization of outliers into the good, the bad and the ugly, distinguishing ugly ones that sabotage ML classifiers; (2) rules to detect different types of outliers by embedding ML models and statistical/loss functions as predicates, to improve the accuracy and explainability, (3) an approach for rule discovery based on meta learning, and (4) a method to fix outliers with right values, instead of simply removing tuples.

We discuss related work in Section 7 and future work in Section 8.

2 OUTLIER MANAGEMENT RULES

This section introduces Outlier Management Rules (OMRs). We start with preliminaries (Section 2.1). We then define OMRs and illustrate how they distinguish between different types of outliers, as well as how they enhance and interpret ML predictions (Section 2.2).

2.1 Preliminaries

We start with basic notations and concepts.

Datasets. Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where R_i is a relation schema $R(A_1, \ldots, A_k, Y)$ and each A_i is an attribute (*a.k.a.* feature). A dataset $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_i is a relation of R_i. Each tuple in D_i has a categorical label Y, for supervised classification model training. The active domain of attribute A_i in D_j, denoted as $dom(A_i)$, refers to discrete values or continuous ranges of A_i that are valid inputs for the ML classifier. Note that our method can be applied to any tabular data format, such as TSV, CSV or Excel, which supports various ML applications.

Accuracy. Following the closed-world assumption [75], we denote by $\mathrm{acc}(\mathcal{M}, \mathcal{D})$ the accuracy of an ML classifier $\mathcal{M}$ that is trained and evaluated with splits of a dataset $\mathcal{D}$. We define $\mathrm{acc}(\mathcal{M}, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(\hat{y}_i = y_i)$, where $\mathcal{D}$ contains n samples s_i ($i \in [1, n]$), each s_i consists of a true label y_i and features X_i, $\hat{y}_i$ is the predicted label of s_i, and $\mathbb{I}(\cdot)$ is the function that returns 1 if true and 0 otherwise. We focus on classification tasks in this paper, where the label is typically categorical and serves as the target for training classifiers.

Influential features. We say that attribute A_i is *influential* if $\mathrm{acc}(\mathcal{M}, \mathcal{D}) - \mathrm{acc}(\mathcal{M}, \mathcal{D}\backslash A_i) > \alpha$, where α is a threshold, *i.e.,* A_i significantly affects the prediction accuracy of $\mathcal{M}$ on dataset $\mathcal{D}$.

Imbalanced features. We say that a feature/attribute A_j is *imbalanced* if the proportion of the most common value of A_j attribute exceeds that of the least common one by a given threshold β.

2.2 OMRs with ML and Function Predicates

The objective of OMRs is twofold: (a) to detect outliers in $\mathcal{D}$ and classify them into good, bad, and ugly based on their impact on $\mathcal{M}$, as indicated in Example 1; and (b) to fix ugly outliers with right values. Below we first define the predicates of OMRs.

Predicates. Over a database schema $\mathcal{R}$, a dataset $\mathcal{D}$ of $\mathcal{R}$, and an ML classifier $\mathcal{M}$, the predicates of OMRs have the following forms:

$$\begin{aligned} p \quad ::= \quad & R(t) \mid t.A \otimes c \mid t.A \otimes s.B \mid \mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) = \tau \mid \\ & F(t, A, \mathcal{D}, \mathcal{M}) = \tau \mid \mathrm{Rk}(t.A), \end{aligned}$$

where $\otimes$ is one of $=, \neq, <, \leq, >, \geq$, and τ is true or false. We write $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) =$ true simply as $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ when it is clear in the context; similarly for $F(t, A, \mathcal{D}, \mathcal{M}) =$ true. We refer to $R(t), t.A \otimes c, t.A \otimes s.B$ as *logic predicates.* Following tuple relational calculus [6], (1) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is called a tuple variable *bounded by* $R(t)$. (2) When t is bounded by $R(t)$ and A is an attribute of R, $t.A$ denotes the A-attribute of t. (3) In $t.A \otimes c$, c is a constant in $dom(A)$. (4) In $t.A \otimes s.B$, A and B have the same type. (5) Rk is an indicator for good, bad and ugly.

In addition to logic predicates, OMRs support the following *ML predicates* $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ and *function predicates* $F(t, A, \mathcal{D}, \mathcal{M})$.

ML predicates. We consider two types of ML predicates $\mathcal{M}_o$.

(1) Detectors. OMRs leverage pre-trained ML-based detectors to identify outliers and influential features, such as the following:

- $\mathcal{M}_{\mathrm{DetO}}(t, A, \mathcal{D})$, an existing ML classifier that returns true if $t.A$ is an outlier, and false otherwise, *e.g.,* unsupervised [21, 110, 128, 138, 157, 184] and label-based (semi-supervised, supervised and rule-based) [34, 37, 91, 130, 131, 137, 139, 148, 183] detectors.

- $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$, an ML classifier for determining whether the A-attribute of relation schema R is an influential feature for the given ML classifier $\mathcal{M}$. It returns true if so, and false otherwise.

(2) Fixes to outliers. OMRs also employ ML models to determine fixes to ugly outliers via distribution analysis and data extraction.

- $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$, where $t.A$ is an ugly outlier (either attribute or label), and v is a candidate value for $t.A$. It returns true if v is a fix to $t.A$ with a high confidence, where v is determined by, *e.g.,* a regression model that can learn attribute distribution, association analysis of attribute $t.A$ and other attributes of tuple t, or a classification model for labels and categorical features.
- $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$, where $t.A$ and v are the same as above. It returns true if v is a fix to $t.A$ with a high confidence, where v is extracted from an external data source KB.

The ML and statistical models learn column distributions and attribute association from the normal data, predict values or extract correct external data for the replacement of ugly outliers.

Function predicates. OMRs may embed statistical and loss functions F as predicates, to determine whether $t.A$ is an outlier, whether $t.A$ triggers substantial loss, or whether a dataset is imbalanced.

- $\mathsf{outlier}(D, R, t, A, \theta)$, where R is a relation schema in $\mathcal{R}$, D is an instance of R, A is an attribute of R, t is a tuple of D, and θ is a hyperparameter. It returns true if the value $t.A$ differs from $s.A$ for at least a factor θ of all tuples s in D, and false otherwise.
- $\mathsf{loss}(\mathcal{M}, D, t, A)$, which represents the loss function of classifier $\mathcal{M}$ trained on dataset D, where t is a tuple and A is an attribute of D. For example, it can be the hinge loss for SVM classifiers [23]. We use $\mathsf{loss}(\mathcal{M}, D, t, A) \leq \lambda$ and $\mathsf{loss}(\mathcal{M}, D, t, A) > \lambda$ to check if the loss is below threshold λ. Ugly outliers struggle to reduce their loss during the model training, while good and bad outliers tend to converge to small values, thus improving $\mathcal{M}$'s accuracy.
- $\mathsf{imbalanced}(D, R, t, A, \delta)$, where R, D, A, t are as above, and δ is a configurable parameter. It returns true if the number of tuples in D grouped by $t.A$ is smaller than the counts of the other groups by A values, by at least a factor δ, and false otherwise.

We show how these functions are implemented in [1].

Outlier Management Rules. For a database schema $\mathcal{R}$, a dataset $\mathcal{D}$ of $\mathcal{R}$, and an ML classifier $\mathcal{M}$, an OMR φ over $\mathcal{R}$ has the form of

$$\varphi = X \longrightarrow p_0,$$

where X is a conjunction of predicates over $\mathcal{R}$, $\mathcal{M}$ and $\mathcal{D}$, and p_0 is a predicate such that all its tuple variables are bounded in X. We refer to X and p_0 as the *precondition* and *consequence* of φ, respectively.

We consider OMRs for a given ML classifier $\mathcal{M}$ and a dataset $\mathcal{D}$ used for training $\mathcal{M}$. We will then see real-life OMRs learned from Diabetes in Table 1 and Apple [48], and illustrate how they can detect (ugly) outliers, improve and explain ML models.

(1) Catching outliers. We use OMRs of the form $X \longrightarrow \mathrm{Rk(t.A)}$ to classify outlier $t.A$ as good, bad or ugly. Below are some examples.

(a) $\varphi_1 = R_{\mathsf{A}}(t) \land \mathcal{M}_{\mathsf{DetO}}(t, \mathsf{Quality}, \mathsf{Apple}) \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Size}, 0.30) \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Sweetness}, 0.76) \land \mathsf{loss}(\mathcal{M}, \mathsf{A}, t, \mathsf{Quality}) \leq 0.52 \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Size}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Sweetness}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Quality}, \mathcal{M}) \longrightarrow \mathsf{good}(t.\mathsf{Quality})$. Specifically, in the Apple dataset of schema R_{A}, $\mathsf{loss}(\cdot)$ is the hinge loss function. It says that while t.Quality is an outlier (by $\mathcal{M}_{\mathsf{DetO}}(\cdot)$), it has positive impact on the SVM classifier $\mathcal{M}$. Intuitively, t.Size and t.Sweetness are identified as feature outliers by $\mathsf{outlier}(\cdot)$ and influential by $\mathcal{M}_{\mathsf{DetI}}(\cdot)$; moreover, they effectively contribute to the training of $\mathcal{M}$; as a consequence, when training $\mathcal{M}$, $\mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality})$ can be reduced below threshold $\lambda = 0.52$ on the label t.Quality. Hence tuple t with the feature and label outliers benefit model $\mathcal{M}$.

(b) $\varphi_2 = R_{\mathsf{A}}(t) \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Ripeness}, 0.10) \land \mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality}) \leq 0.52 \land \mathsf{imbalanced}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Ripeness}, 0.30) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Ripeness}, \mathcal{M}) = \mathsf{false} \longrightarrow \mathsf{bad}(t.\mathsf{Ripeness})$. It says that t.Ripeness is an outlier, but it does not harm $\mathcal{M}$'s training since $\mathsf{loss}(\cdot) \leq 0.52$. Intuitively, outlier t.Ripeness is not influential (by $\mathcal{M}_{\mathsf{DetI}}(\cdot) = \mathsf{false}$); it is an occasional erroneous value since both $\mathsf{outlier}(\cdot)$ and $\mathsf{imbalanced}(\cdot)$ are true. Hence it is bad but not ugly.

(c) $\varphi_3 = R_{\mathsf{A}}(t) \land \mathsf{imbalanced}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Juiciness}, 0.2) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Size}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Sweetness}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(R_{\mathsf{A}}, \mathsf{Juiceness}, \mathcal{M}) \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Size}, 0.30) = \mathsf{false} \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Juiciness}, 0.16) \land \mathsf{outlier}(\mathsf{Apple}, R_{\mathsf{A}}, t, \mathsf{Sweetness}, 0.76) = \mathsf{false} \land \mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality}) > 0.52 \longrightarrow \mathsf{ugly}(t.\mathsf{Juiciness})$. It states that t.Juiceness is an ugly outlier, as it is an outlier, influential and imbalanced; the loss of $\mathcal{M}$ on tuple t constantly exceeds threshold $\lambda = 0.52$, and $\mathcal{M}$ cannot converge on t. Moreover, other influential features t.Size and t.Sweetness are not outliers ($\mathsf{outlier}(\cdot) = \mathsf{false}$).

(d) OMRs can detect ugly outliers that arise from combinations of tuples, even when the individual tuples appear normal in isolation. Consider $\varphi_4 = R_{\mathsf{Diabetes}}(t_x) \land R_{\mathsf{Diabetes}}(t_y) \land R_{\mathsf{Diabetes}}(t_z) \land \mathcal{M}_{\mathsf{DetI}}(\mathsf{Diabetes}, \mathsf{Glucose}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(\mathsf{Diabetes}, \mathsf{HbA1c}, \mathcal{M}) \land \mathcal{M}_{\mathsf{DetI}}(\mathsf{Diabetes}, \mathsf{InsulinResistance}, \mathcal{M}) \land t_x.\mathrm{HbA1c} > t_y.\mathrm{HbA1c} \land t_x.\mathrm{InsulinResistance} > t_z.\mathrm{InsulinResistance} \land \mathcal{M}_{\mathsf{DetO}}(t_x, \mathsf{Glucose}, \mathsf{Diabetes}) \land t_y.\mathrm{PredictedLabel} = 1 \land t_z.\mathrm{PredictedLabel} = 1 \longrightarrow \mathsf{ugly}(t_x)$. We substitute tuples t_8, t_7, and t_6 from Table 1 for variables t_x, t_y, and t_z in φ_4. Model $\mathcal{M}$ individually predicts t_6, t_7, and t_8 as diabetes, diabetes, and normal by medical criteria.

However, when $\mathcal{M}$ is trained on t_6, t_7 and t_8 together, it misclassifies t_8 as diabetic due to similarities in key diagnostic features (Glucose, HbA1c, InsulinResistance). Tuple t_8 has the highest Glucose, slightly higher HbA1c than t_7, and slightly higher InsulinResistance than t_6. Since $\mathcal{M}$ predicts t_6 and t_7 as diabetic, it wrongly infers the same for t_8, despite it not meeting ADA criteria. Thus, t_8 is an ugly outlier caused by misleading patterns across the tuple combination. This tricky ugly outlier is caught by OMR φ_4.

(2) Fixing ugly outliers. We use OMRs of the forms $X \longrightarrow t.A = c$ or $X \longrightarrow t.A = s.B$ to fix ugly outlier $t.A$ with value c or $s.B$, which are determined by means of ML predicates $\mathcal{M}_{\mathsf{FixA}}$ or $\mathcal{M}_{\mathsf{FixE}}$, and logic reasoning in precondition X. We defer the details to Section 5.

(3) Improving existing ML detectors. We use OMRs of the form $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) \land X_1 \longrightarrow p_0$ to improve the accuracy of ML model $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ by incorporating logic conditions X_1. That is, while $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ predicts true (resp. false), φ override the decision if condition X_1 is satisfied (resp. violated), to reduce FPs (resp. FNs) of $\mathcal{M}_o$. Here $\mathcal{M}_o$ can be $\mathcal{M}_{\mathsf{DetO}}$, $\mathcal{M}_{\mathsf{DetI}}$, $\mathcal{M}_{\mathsf{FixA}}$ or $\mathcal{M}_{\mathsf{FixE}}$.

Consider OMR φ_5 from Diabetes: $R_{\mathsf{Diabetes}}(t) \land \mathcal{M}_{\mathsf{DetI}}(\mathsf{Diabetes}, \mathsf{BMI}, \mathcal{M}) = \mathsf{false} \land X_1 \longrightarrow \mathcal{M}_{\mathsf{DetI}}(\mathsf{Diabetes}, \mathsf{BMI}, \mathcal{M}) = \mathsf{true}$, where $\mathcal{M}$ is the XGBoost classifier, and X_1 is $t.\mathsf{BMI} \leq 21 \land t.\mathsf{InsulinResistance} \geq 8.0$. While ML detector $\mathcal{M}_{\mathsf{DetI}}(\cdot)$ classifies

BMI as not influential, if tuple t satisfies X_1, the prediction should be flipped. Indeed, X_1 reveals the pathogenesis of lean diabetes [165], where patients exhibit high insulin resistance $\geq$ 8.0) alongside low BMI ($\leq$ 21). BMI is typically considered a non-critical indicator under ADA guidelines [14], but in this context, it becomes an influential diagnostic feature ($\mathcal{M}_{\text{DetI}}(\text{Diabetes}, \text{BMI}, \mathcal{M}) = \text{true}$).

(4) Explaining ML predictions. We use OMRs of the form $X \to \mathcal{M}_{\text{DetO}}(t, A, \mathcal{D})$ to explain predictions of ML detector $\mathcal{M}_{\text{DetO}}$, by discovering precondition X to explain why $\mathcal{M}_{\text{DetO}}(t, A, \mathcal{D})$ holds.

Consider OMR φ_6 learned from Apple: $R_{\text{A}}(t) \wedge \text{outlier}(\text{Apple}, \mathcal{R}_A, t, \text{Juiciness}, 0.16) \wedge \text{imbalanced}(\text{Apple}, R_A, t, \text{Juiciness}, 0.2) \wedge \mathcal{M}_{\text{DetI}}(\mathcal{R}_A, \text{Juiceness}, \mathcal{M}) \to \mathcal{M}_{\text{DetO}}(t, \text{Juiceness}, \mathcal{D})$. The OMR explains why the ML detector $\mathcal{M}_{\text{DetO}}(\cdot)$ identifies t.Juiciness as an outlier. It is because (a) the value of t.Juiciness differs significantly from the other tuples ($\text{outlier}(\text{Apple}, R_A, t, \text{Juiciness}, 0.16)$), (b) Juiciness is influential for $\mathcal{M}$ by $\mathcal{M}_{\text{DetI}}(\cdot)$, and (c) t.Juiciness makes the dataset imbalanced ($\text{imbalanced}(\text{Apple}, R_A, t, \text{Juiciness}, 0.2)$).

Semantics. Consider an instance $\mathcal{D}$ of database schema $\mathcal{R}$. A *valuation* of tuple variables of an OMR $\varphi = X \to p_0$ in $\mathcal{D}$, or simply *a valuation of* φ, is a mapping h that maps each t in relation atom $R(t)$ of φ to a tuple in the relation of schema R in $\mathcal{D}$.

We say that h satisfies a predicate p, denoted as $h \models p$, if h satisfies the following conditions. (1) If p is a relation atom $R(t)$, $t.A \otimes c$ or $t.A \otimes s.B$, then $h \models p$ is interpreted as in tuple relational calculus [6]. (2) If p is an ML predicate $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$, then $h \models p$ if $\mathcal{M}_o$ predicts true on tuple $t \in \mathcal{D}$ and its attribute A *w.r.t.* the given ML classifier $\mathcal{M}$; similarly for $p = F(t, A, \mathcal{D}, \mathcal{M})$.

For a set X of predicates p, we write $h \models X$ if $h \models p$ for all predicates $p \in X$. For an OMR $\varphi = X \to p_0$, we write $h \models \varphi$ such that if $h \models X$, then $h \models p_0$. An instance $\mathcal{D}$ of $\mathcal{R}$ satisfies φ if $h \models \varphi$ for all valuations h of φ in $\mathcal{D}$, denoted by $\mathcal{D} \models \varphi$. We say that $\mathcal{D}$ satisfies a set Σ of φ, written by $\mathcal{D} \models \Sigma$, if $\mathcal{D} \models \varphi$ for all $\varphi \in \Sigma$.

Remark. These OMRs differ from previous data quality rules like DCs [12], MDs [52], and REEs [56, 57], since (1) DCs and MDs do not support ML models like $\mathcal{M}_{\text{DetO}}(t, \text{Quality}, \mathcal{D})$; and (2) REEs do not support functions like $\text{loss}(\mathcal{M}, D, t) \otimes \lambda$. Unlike prior rules, OMRs target outliers to enhance ML classifier accuracy by reducing false positives/negatives through logical predicates; they incorporate ML-based outlier detectors and statistical/loss functions to capture insights of downstream models. Thus, OMRs require different discovery methods than those for previous rules (see Section 4). Moreover, OMRs only manage and improve the training data, and do not repair the classification outputs of ML models.

3 OHUNT: OUTLIER HUNTER

This section outlines OHunt, a system for detecting and fixing ugly outliers. Given a schema $\mathcal{R}$, dataset $\mathcal{D}$, and classifier $\mathcal{M}$, OHunt first learns a set of OMRs Σ from $\mathcal{D}$ guided by $\mathcal{M}$, and then recursively applies them to detect and repair ugly outliers. The cleaned training data is used to retrain $\mathcal{M}$. OHunt consists of two main modules.

(1) Rule discovery. This module targets the following problem.

- *Input*: $\mathcal{R}, \mathcal{D}, \mathcal{M}$ as above, a set $\mathcal{P}$ of potential OMR predicates, objective function $f_d(\cdot)$ (resp. $f_f(\cdot)$) for detection accuracy (resp. accuracy improvement of $\mathcal{M}$; see Section 4 for $\mathcal{P}, f_d(\cdot), f_f(\cdot)$).
- *Output:* A set $\Sigma = \Sigma_d \cup \Sigma_f$ of OMRs, where Σ_d is the set of OMRs mined for detecting (ugly) outliers, and Σ_f is the set of OMRs mined for fixing ugly outliers detected by Σ_d.

We propose the OLeaner algorithm for learning Σ in Section 4, with the following properties: (1) It frames OMR discovery as a meta-learning problem, optimizing for accuracy improvement and avoiding exponential search in rule discovery. (2) Bayesian optimization is used to find hyperparameters that enhance ugly outlier detection and improve $\mathcal{M}$'s classification accuracy.

(2) Outlier correction. The module focuses on the problem below.

- *Input*: $\mathcal{R}, \mathcal{D}, \mathcal{M}$ as above, and the set Σ of OMRs learned.
- *Output*: Dataset $\mathcal{D}_c$ by fixing ugly outliers in $\mathcal{D}$ with Σ.

OFix detects ugly outliers in $\mathcal{D}$ using OMRs from Σ_d and fixes them with OMRs from Σ_f in Section5. Specifically, it chases $\mathcal{D}$ with Σ_f[149], accumulating a set Γ of ground truth and referencing it during the process. The fixes are guaranteed correct if Σ_f and Γ are accurate, along with the ML and function predicates in rules.

4 DISCOVERING OMRS BY META LEARNING

This section develops algorithm OLeaner for learning OMRs.

Challenges. Rules are traditionally mined by levelwise enumeration [54, 68, 86, 132, 159]. However, the approach does not suffice for OMRs. (1) It cannot determine hyperparameters, such as optimal θ in $\text{outlier}(D, R, t, A, \theta)$, especially for continuous values. (2) Rule quality metrics like support and confidence are invalid since OMRs are evaluated based on their impact on $\mathcal{M}$'s accuracy. For example, OMRs may have low support but improve accuracy on rare outliers. (3) Classical rule mining finds all rules above threshold, while we focus on OMRs that enhance detection and classification accuracy for $\mathcal{M}$. (4) OMRs support multi-variable rules across tables, unlike current discovery algorithms, such as those for DCs.

To address these challenges, we model OMR discovery as a meta-learning problem and apply Bayesian optimization for efficient search. We first review model-evaluation-based meta learning (Section 4.1), and then present OLeaner for OMR discovery (Section 4.2).

4.1 Meta Learning based on Model Evaluation

Meta-learning involves learning new tasks by observing ML model performance on existing tasks [168]. It takes as input: (a) a set of known tasks $t_j \in T$; (b) a set of learning algorithms with configurations $\theta_i \in \Theta$; (c) the set $\mathbf{P}$ of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j measured by a predefined measure (*e.g.,* accuracy) and model evaluation technique (*e.g.,* cross validation); and (d) evaluations $\mathbf{P}_{\text{new}}$ for a new task t_{new}. Meta-learning trains a meta-learner L to predict optimal configurations Θ^*_{new} for t_{new} based on the meta-data $\mathbf{P} \cup \mathbf{P}_{\text{new}}$ [168].

Various meta-learning methods accelerate and improve hyperparameter selection of ML models in a data-driven way, classified as ranking-based [4, 25, 40, 105, 105], configuration space design [87, 136, 176], and configuration transfer [60, 66, 69, 160, 163, 177].

4.2 OMR Discovery as Meta Learning Problem

As shown in Algorithm 1, given a consequence predicate p_0 (Section 2), OLeaner for OMRs discovery is to find a combination of predicates along with their hyperparameters for precondition X, such that each $\varphi = X \to p_0$ either detects or fixes outliers.

Formulation. The discovery of OMR φ can be formulated as a meta learning problem as follows. The tasks $t_j \in T$ are either distinguishing outliers or fixing ugly ones in dataset $\mathcal{D}$ (line 1). The learning algorithms in meta-learning can be parametrized as: $\mathbb{S}(p_1, x_1) \wedge \mathbb{S}(p_2, x_2) \wedge \ldots \wedge \mathbb{S}(p_n, x_n) \rightarrow p_0$, where $p_1, \ldots, p_n$ are predicates of OMRs, and $\mathbb{S}(p, x)$ is a selection function defined as:

$$\mathbb{S}(p, x) = \begin{cases} p, & x = 1 \\ \text{true}, & x = 0 \end{cases}$$

Thus, the configurations $\theta_i \in \Theta$ of a learning algorithm include the set $\{x_1, x_2, \ldots, x_n\}$ that controls whether each predicate is selected in precondition X, and hyperparameters in predicates, *e.g.,* the threshold λ in $\text{loss}(\mathcal{M}, D, t, A) \leq \lambda$. The set $\mathbf{P}$ of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j is measured by the accuracy $f_d(\cdot)$ of outlier detection or the improvements $f_f(\cdot)$ of downstream classifiers (line 2). Based on this formulation, OLeaner trains a meta-learner L that recommends configurations Θ^* for tasks $t_j \in T$ based on the evaluation $\mathbf{P}$ (initialized as $\emptyset$). The predicted Θ^* specifies which predicates are selected in precondition X, and what values should be assigned to hyperparameters of those selected predicates, to make a useful OMR $X \rightarrow p_0$.

Objective functions $f_d(\cdot)$ and $f_f(\cdot)$ are defined as follows. (1) For $f_d(\cdot)$, we train classifier $\mathcal{M}$ on dataset $\mathcal{D}$ and identify potential ugly outliers U_1 as samples with loss values exceeding a threshold λ (*e.g.,* initialized as 1 for hinge loss), which is refined during optimization. For an OMR φ_1, let U_2 be the set of ugly outliers it detects. Then $f_d(\varphi_1) = \frac{TP+TN}{|\mathcal{D}|}$, where $TP = |U_1 \cap U_2|$ is the number of correctly identified ugly outliers, and $TN = |\mathcal{D}| - |U_1 \cap U_2| - |U_1 \setminus U_2| - |U_2 \setminus U_1|$ is the number of correctly identified non-ugly samples. (2) To define $f_f(\cdot)$ for a fixing OMR φ_2, we apply φ_2 to repair ugly outliers in $\mathcal{D}$, yielding a cleaned dataset $\mathcal{D}_\varphi$. We retrain $\mathcal{M}$ on $\mathcal{D}_\varphi$ and set $f_f(\varphi_2) = \text{acc}(\mathcal{M}, \mathcal{D}_\varphi) - \text{acc}(\mathcal{M}, \mathcal{D})$. Given $f_d(\cdot)$ or $f_f(\cdot)$ for task t_j, the meta-learner L returns optimized configurations Θ^*, where each $\theta_i \in \Theta^*$ defines selected predicates and hyperparameters for OMRs.

Algorithm. We employ Bayesian optimization [10] as the meta learning algorithm for OMRs, which initializes hyperparameter configurations $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ by domain experts or randomly initialized (line 3). Based on Θ, it evaluates the target metric (*e.g.,* detection accuracy) on each task t_j (*e.g.,* detecting ugly outliers), and initializes the sets $\mathbf{P}$, Σ, Σ_d and Σ_f as empty sets (line 4).

Then the optimization for t_j iterates as follows. A surrogate model is employed to model the distribution of target metric values based on $\mathbf{P}$, which facilitates the Bayesian optimization to determine the next round hyperparameter configuration θ_{m+1} by selecting the point that maximizes the acquisition function [64, 155]. This further updates both $\mathbf{P} = \mathbf{P} \cup \{P(\theta_{m+1}, t_j)\}$ and the distribution of surrogate model guided by the target metric values. This iteration returns its optimal hyperparameter configuration Θ^* that achieves the highest target metric on task t_j, when the budget limit (*e.g.,* number of evaluations, runtime) is reached (lines 5-10).

When the configurations Θ are in place, OLeaner learns the predicates of rules in Σ_d and Σ_f, and determines OMRs (line 11). Specifically, the OMRs in Σ_f and Σ_d are learned by solving the meta-learning problem modeled above with the following procedures. (1) OLeaner selects predicates for $\varphi = X \rightarrow p_0$. For each configuration $\theta_i \in \Theta$, we select predicates $p_1, p_2, \ldots, p_n$ and their combinations from θ_i using $\mathbb{S}(p, x)$. For each $x_k \in x$, which

Algorithm 1: Algorithm OLeaner

```
Input: $R$, $\mathcal{D}$, $\mathcal{M}$, $\mathcal{P}$, $f_d(\cdot)$ and $f_f(\cdot)$ as stated in Section 3.
Output: A set of detecting and fixing OMRs $\Sigma = \Sigma_d \cup \Sigma_f$
1  Define $T$ as outlier detection and fixing tasks;
2  Define $f_d(\cdot)$ and $f_f(\cdot)$ as detecting and fixing evaluation metrics;
3  Initialize configurations $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ for hyperparameters;
4  Initialization of evaluations and OMR sets: $\mathbf{P} \leftarrow \emptyset$; $\Sigma, \Sigma_f, \Sigma_d \leftarrow \emptyset$;
5  foreach task $t_j \in T$ do
6      while not reaching budget limit do
7          Use Bayesian optimization to select configurations from $\Theta$;
8          Evaluate configuration $\theta_i$ based on function $f_d(\cdot)$ or $f_f(\cdot)$;
9          Update prior scalar evaluations: $\mathbf{P} \leftarrow \mathbf{P} \cup \{P(\theta_i, t_j)\}$;
10         Optimize the acquisition function using the surrogate
             model to determine the next $\theta_{m+1}$ and adds it to $\Theta$ ;
11 foreach configuration $\theta_i \in \Theta$ do
12     Identify $\varphi : X \rightarrow p_0$ based on configuration $\theta_i$ using $\mathbb{S}(p, x)$;
13     Use $\varphi$ to evaluate the objective function $f_d(\cdot)$ or $f_f(\cdot)$ ;
14     if the objective function $f_d(\cdot)$ or $f_f(\cdot)$ is enhanced then
15         Include the rule $\varphi : X \rightarrow p_0$ in $\Sigma_d$ or $\Sigma_f$;
16 Return the set of discovered OMRs: $\Sigma \leftarrow \Sigma_d \cup \Sigma_f$
```

is $x = \{x_1, x_2, \cdots, x_n\}$, OLeaner selects predicate $p_{i,k}$ from θ_i if $x_k = 1$, and ignores it otherwise. Thus, we obtain OMR $\varphi : X \rightarrow p_0$ using θ_i (line 12). (2) OLeaner applies φ to detect (resp. fix) ugly outliers, and evaluates the accuracy using $f_d(\cdot)$ (resp. $f_f(\cdot)$) (line 13). If the objective function $f_d(\cdot)$ (resp. $f_f(\cdot)$) is enhanced after applying φ, OLeaner adds φ to the set Σ_d (resp. Σ_f) (lines 14-15). Since the training of $\mathcal{M}$ and the computation of U_1 and U_2 typically have a cost of $O(|\mathcal{D}|)$, $f_d(\cdot)$ takes linear time to train. (3) After all iterations of Θ, OLeaner returns $\Sigma = \Sigma_d \cup \Sigma_f$ (line 16). The predicates in precondition X of OMRs $\varphi : X \rightarrow p_0$ in Σ_f are limited to logic predicates, $\mathcal{M}_{\text{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB})$, while those in Σ_d are logic predicates, ML detectors and function predicates. The consequence p_0 of OMRs in Σ_f fixes ugly outliers $t.A$; and p_0 of those in Σ_d is Rk(t.A) classifies $t.A$ as good, bad or ugly.

Example 2: In the meta-learning framework for Diabetes, two candidate OMRs are initialized with expert-defined hyperparameters:

$\phi_1 = R_{\text{Diabets}}(t_x) \wedge \mathcal{M}_{\text{DetO}}(t_x, \text{Glucose, Diabetes}) \wedge \text{loss}(\text{XGBoost, Diabetes}, t_x, \text{RecordedLabel}) > 1 \wedge \text{max\_depth} = 5,$

$\phi_2 = R_{\text{Diabets}}(t_y) \wedge \mathcal{M}_{\text{FixA}}(\text{Diabetes}, t_y, \text{RecordedLabel}, 1) \wedge \text{loss}(\text{XGBoost, Diabetes}, t_y, \text{RecordedLabel}) > 1.5 \wedge \text{learning\_rate} = 0.01 \wedge \text{n\_estimators} = 50$

where "max_depth", "n_estimators" and "learning_rate" are hyperparameters in XGBoost. We instantiate t_x and t_y with tuples t_1 and t_2 from Table 1. Guided by $f_d(\cdot)$ and $f_f(\cdot)$, Bayesian optimization selects predicates via indicator function x and tunes their hyperparameters. This updates the configurations to:

$\phi_1 = R_{\text{Diabets}}(t_1) \wedge \mathcal{M}_{\text{DetO}}(t_1, \text{Glucose, Diabetes}) \wedge \text{loss}(\text{XGBoost, Diabetes}, t_1, \text{RecordedLabel}) > 3 \wedge \text{max\_depth} = 10 \wedge \text{imbalanced}(t_1, \text{Glucose, Diabetes}),$

$\phi_2 = R_{\text{Diabets}}(t_2) \wedge \mathcal{M}_{\text{FixA}}(\text{Diabetes}, t_2, \text{RecordedLabel}, 1) \wedge \text{loss}(\text{XGBoost, Diabetes}, t_2, \text{RecordedLabel}) > 3 \wedge \text{n\_estimators} = 100 \wedge \text{learning\_rate} = 0.05,$

This optimization refines the loss thresholds and XGBoost parameters (*e.g.,* depth, learning rate, estimator count), and incorporates imbalanced$(\cdot)$ to better isolate ugly outliers, enhancing post-repair classification performance on the Diabetes dataset. □

Remark. (1) We model rule discovery as a meta learning problem to avoid the exponential enumeration and find effective OMRs within limited computation cost. The number n of candidate predicates for precondition X is bounded by $\mathcal{D}$'s active domain, with continuous ranges discretized into fixed intervals, and n grows linearly with $|\mathcal{D}|$. Prior research [24, 42, 73, 123, 129, 166, 185] and our experiments show most useful predicates in X are ML and function predicates as shown in Section 2.2, which further reduces candidate size. (2) We use Bayesian optimization for meta learning to discover OMRs because (a) gradient-descent-based methods [13, 61, 126] are unsuitable due to non-differentiable discrete hyperparameters, and (b) Bayesian optimization converges faster [29, 90, 178] than other gradient-free methods [82, 95, 125], by using prior information.

Complexity. OLeaner takes $O(l \times T + I \times (m^3 + K))$ time, where l is the size of the initial configuration set, T is the time to evaluate a configuration, I is the number of iterations, m is the number of attributes in $\mathcal{D}$, and K is the number of steps for optimizing the acquisition function (see [1] for more details).

5 FIXING THE UGLY OUTLIERS

This section develops the algorithm of OHunt for detecting and fixing ugly outliers, denoted by OFix, by employing OMRs. We first present a combination of strategies for finding right values for ugly outliers (Section 5.1). Based on the strategies, we then present OFix to "deep clean" outliers, with accuracy guarantees (Section 5.2).

5.1 Finding Right Values for Ugly Outliers

OFix takes as input a dataset $\mathcal{D}$ of schema $\mathcal{R}$, an ML classifier $\mathcal{M}$ to be trained with $\mathcal{D}$ and a set $\Sigma = \Sigma_d \cup \Sigma_f$ of learned OMRs (Section 4). It detects and fixes ugly outliers of $\mathcal{D}$ by employing OMRs of Σ, to obtain a cleaned dataset $\mathcal{D}_c$ as training data for $\mathcal{M}$.

As shown in Section 2.2, we apply OMRs of the form $X \to \text{ugly}(t.A)$ in Σ_d to identify ugly outliers. The question is how to fix the detected ugly outliers with right values? Prior work [2, 3, 58, 59, 71, 101–103, 115] often remove tuples that contain outliers. As remarked earlier, this approach may change data distribution and variable association, thus reducing $\mathcal{M}$'s classification accuracy.

OFix proposes to fix outliers caught by OMRs $X \to \text{ugly}(t.A)$ with a combination of three strategies as follows. It employs OMRs $X \to t.A = c$ or $X \to t.A = s.B$ in Σ_f to fix $t.A$ with a right value.

(1) Logic deduction. We can deduce value c for $t.A$ by logic reasoning. An example is $R_{\text{Diabetes}}(t) \wedge \text{ugly}(t.\text{RecordedLabel}) \wedge t.\text{HbA1c} \geq 6.5\% \wedge t.\text{InsulinResistance} \geq 5.0 \to t.\text{RecordedLabel} = 1$. If RecordedLabel and HbA1c are high enough, then the patient should be diagnosed with diabetes. This OMR fixes a label outlier.

Logic reasoning helps fix ugly outliers in features or labels. The fixes can be explained. However, this strategy often covers only a few cases in practice. To fix the other cases, we use ML models.

(2) Statistical analyses. We train an ML model $\mathcal{M}_{\text{FixA}}$ to find a value v for ugly outliers $t.A$. Then we can apply OMRs of the form $R(t) \wedge \text{ugly}(t.A) \wedge \mathcal{M}_{\text{FixA}}(\mathcal{D}, t, A, v) \to t.A = v$ to fix $t.A$.

(a) Upon the availability of normal tuples relevant to ugly outliers in $\mathcal{D}$, we train $\mathcal{M}_{\text{FixA}}$ to learn data distribution from these tuples and use it to fix ugly outliers in $\mathcal{D}$. Consider the contextual connection $C = (\mathcal{M}_{N\to U}, U, N)$, where $\mathcal{M}_{N\to U}$ identifies the set N of normal neighbors for the set U of ugly outliers. Consider ugly outliers in influential features or labels, while other features follow the correct data distribution. Then U and N share a similar distribution. To fix the ugly outliers, we train a KNN model $\mathcal{M}_{\text{knn}}$ on non-ugly tuples, setting the feature values in U as *nan*, and use $\mathcal{M}_{\text{knn}}$ to replace the *nan* values with the mean of neighbors' values. Moreover, $\mathcal{M}_{\text{FixA}}$ explores the association between $t.\bar{B}$ (attributes excluding A) and label $t.A$. For instance, a rich area is often associated with high income, and a female name is linked to a girl. We fix ugly outliers $t.A$ by the association between $t.A$ and $t.\bar{B}$ with high confidence [72].

(b) When training $\mathcal{M}_{\text{FixA}}$ is not feasible due to insufficient samples, we use statistical methods. We identify influential features A and the set U of ugly outliers. We compute statistical metrics (*e.g.,* mean or median) for normal tuples under A, and fix outliers in U with these statistical values. For ugly label outliers $t.Y$, we identify normal tuples with similar features and assign the most frequent label to $t.Y$.

(3) Value extraction from external sources. We train another ML model $\mathcal{M}_{\text{FixE}}$ to predict v using an external data source KB with overlapping information. Then we use OMRs $R(t) \wedge \text{ugly}(t.A) \wedge \mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB}) \to t.A = v$ to fix outliers $t.A$.

For instance, when KB is a knowledge base, we align tuples t in $\mathcal{D}$ and vertices v in KB referring to the same entity, *e.g.,* via HER [55]. Model $\mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB})$ fixes ugly outliers $t.A$ using features of v. KB can be extracted from sources like Wikidata [172] and Kaggle [15] via query interfaces (e.g., Wikidata's SPARQL [172], Kaggle's [15] and GraphQL [135] API), with HER enhancements [16, 79, 187]. Referencing KB is optional. If available, OFix mitigates noise in both KB and $\mathcal{D}$; otherwise, correction is data-driven.

For example, consider the ugly outlier ugly(t.Quality) detected by φ_3 on Apple (Section 2.2). Then OMR $R_A(t) \wedge \text{ugly}(t.\text{Quality}) = -5.96 \wedge \mathcal{M}_{\text{FixE}}(\text{Apple}, t, \text{Quality}, -3.25, \text{Sales}) \to t.\text{Quality} = -3.25$ corrects the outlier value by matching the tuple $t.\text{Quality} = -5.96$ in relation Apple with the vertex $v = -3.25$ in graph Sales.

If different values are suggested for fixing an ugly outlier $t.A$, we pick the one that maximally improves the accuracy of model $\mathcal{M}$.

5.2 A Recursive Algorithm for Fixing Outliers

Based on the strategies, we develop a "deep cleaning" algorithm OFix, to recursively fix ugly outliers, by chasing dataset $\mathcal{D}$ with the set Σ of OMRs learned. Below we first extend the chase [149].

The chase revised. We start with the notion of fixes.

Fixes. To keep track of fixes to ugly outliers, for each attribute $t.A$, we maintain relations $[t.A]_\otimes$, where $\otimes$ ranges over $=, \neq, <, \leq, >, \geq$. Each $[t.A]_\otimes$ consists of attributes $t'[B]$ or constants c such that $t.A \otimes t'[B]$ and $t.A \otimes c$ are either in the ground truth Γ (see below) or deduced during the chase. In particular, if $t.A$ is an outlier, then $[t.A]_=$ may include Rk (*i.e.,* good, bad, ugly) as a special value for label, and a constant in $[t.A]_=$ that makes a fix to $t.A$.

We ensure that each $[t.A]_=$ and $[t.A]_\neq$ are equivalence relations, *i.e.,* reflexive, symmetric and transitive, while each $[t.A]_\otimes$ is reflexive and transitive for the other $\otimes$. Moreover, each $[t.A]_\otimes$ is *valid*, *i.e.,* no c_1 and c_2 in the same $[t.A]_\otimes$ conflict with each other, *e.g.,* $[t.A]_=$ does not include constants c_1 and c_2 where $c_1 \neq c_2$.

Ground truth. The initial set of fixes is denoted as Γ, which are collected and validated by users, domain experts or crowd-sourcing.

The chase. A *chase step* of dataset $\mathcal{D}$ with Σ_f at a set $\mathcal{F}$ of fixes is

$$\mathcal{F} \Rightarrow_{(\varphi,h)} \mathcal{F}'.$$

Here $\varphi : X \rightarrow p_0$ is an OMR in Σ_f and h is a valuation of φ such that (a) all predicates in X are *validated* by $\mathcal{F}$; that is, if p is $t.A \otimes s.B$, then $h(s).B \in [t.A]_\otimes$ for $[t.A]_\otimes$ in $\mathcal{F}$; if p is $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$, then $\mathcal{M}_o$ predicts true at $h(t).A$ and all the data involved in the prediction is in $\mathcal{F}$ (*e.g.,* neighbor features if $\mathcal{M}_o$ is KNN); similarly for $F(t, A, \mathcal{D}, \mathcal{M})$; if p is $\mathsf{Rk}(t.A)$, then Rk is in $[t.A]_\otimes$; and (b) the consequence p_0 extends $\mathcal{F}$ to $\mathcal{F}'$, by adding c (resp. $s.B$, Rk) to $[t.A]_\otimes$ in $\mathcal{F}'$ if p_0 is $t.A \otimes c$ (resp. $t.A \otimes s.B$, $\mathsf{Rk}(t.A)$). That is, the chase expands ground truth with validated fixes in the process.

A *chasing sequence* ξ of $\mathcal{D}$ by (Σ, Γ) is a sequence

$$\mathcal{F}_0, \ldots, \mathcal{F}_n,$$

where $\mathcal{F}_0$ is Γ and for each $i \in [1, n]$, there exist φ in Σ and valuation h of φ such that $\mathcal{F}_{i-1} \Rightarrow_{(\varphi,h)} \mathcal{F}_i$ is a valid chase step, *i.e.,* $\mathcal{F}_i$ is valid.

The sequence ξ *terminates* when no more OMRs in Σ can be applied to further extend $\mathcal{F}_n$ with new fixes. Such a sequence is referred to as a *terminal chasing sequence*. When a sequence terminates, we refer to the set $\mathcal{F}_n$ of fixes as the result of ξ.

Challenges. Directly implementing the chase may lead to conflicts when multiple rules are applicable to fixing the same outlier, with different values. It is also costly if we apply such rules one by one.

Algorithm. Based on the revised chase, we develop OFix as shown in Algorithm 2. It consists of initialization, chasing and correction.

(1) Algorithm OFix first employs OMRs in Σ_d to find all outliers. It stores the ugly ones in U (line 1). Then after trained with all kinds of outliers and normal tuples offline (line 2), $\mathcal{M}_{\mathsf{FixA}}$ and $\mathcal{M}_{\mathsf{FixE}}$ can predict correct value $t.A = v$ to fix $\mathsf{ugly}(t.A)$. The ground truth Γ is initialized with validated facts in $\mathcal{D}$ and high-quality KB (line 3).

(2) During chasing steps, OFix adopts batch processing when multiple OMRs are applicable simultaneously. Denote such rules as the subset $\Sigma_t \subseteq \Sigma_f$ when the same unverified $\mathsf{ugly}(t.A)$ appears in their precondition X (line 7). We initialize the candidate correction set C of Σ_t as $\emptyset$ (line 8). OMRs from Σ_t are activated in parallel to deduce fixes c for $t.A$ in U. If the fix c is already in Γ, $\mathsf{ugly}(t.A)$ is fixed directly using c (lines 10-11). Otherwise, if c is inferred, $\mathsf{ugly}(t.A)$ is added to the set $C = \{c_i \mid \exists\varphi \in \Sigma_t : \varphi \models X \rightarrow c_i\}$ (lines 12-13).

After the batch processing, OFix resolves conflicts in C as follows. When multiple fixes $t.A = c$ are deduced, OFix employs $\mathsf{imp}(\mathcal{M}, c)$ (*i.e.,* $\mathcal{M}$'s classification accuracy improvement) to decide the suitable value, which is computed as $c^* = \mathsf{argmax}_{c \in C}\mathsf{imp}(\mathcal{M}, c)$ (lines 15-16). Then, c^* is used to fix $t.A$, and added to $[t.A]_=$ (*i.e.,* Γ) to resolve conflicts in C (line 17). After $\mathsf{ugly}(t.A)$ is fixed, OFix removes it from U (line 17). The remaining $\mathsf{ugly}(t.A)$ in U and updated Γ activate more OMRs from Σ_f, and the chase continues (lines 4-5). When there are no changes can be made, the chase terminates.

(3) Upon termination, OFix deduces a new training set $\mathcal{D}_c$ from $\mathcal{D}$ by fixing all ugly outliers $t.A$ in U with values in $[t.A]_=$ (line 18).

Algorithm 2: Algorithm OFix

```
Input: $R$, $\mathcal{D}$, $\mathcal{M}$, KB as stated in above, $\Sigma = \Sigma_d \cup \Sigma_f$ of OMRs.
Output: A dataset $\mathcal{D}_c$ by fixing ugly outliers in $\mathcal{D}$ with $\Sigma$.
1  Add all ugly outliers $t.A$ identified by $\Sigma_d$ to $U$;
2  Train $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ offline;
3  $\Gamma \leftarrow \{[t.A] = v \mid t.A \in \mathcal{D}, \mathsf{KB}\}$;  flag := true;
4  while flag do
5      flag := false;
6      foreach ugly outlier $t.A \in U$ do
7          Identify the subset $\Sigma_t$ of rules from $\Sigma_f$ whose
             preconditions involve $\mathsf{ugly}(t.A)$;
8          The candidate correction set $C \leftarrow \emptyset$;
9          foreach OMR $\varphi \in \Sigma_t$ in parallel do
10             if $\varphi$ produces a value $c \in \Gamma$ then
11                 Fix $t.A$ directly using $c$;
12             else if $\varphi$ deduces a new value $c \notin \Gamma$ then
13                 $C \leftarrow \{c_i \mid \exists\varphi \in \Sigma_t : \varphi \models X \rightarrow c_i\}$;
14         if multiple $c_i$'s appear in $C$ and $t.A$ has not been fixed then
15             Compute $\mathsf{imp}(\mathcal{M}, c)$ for each candidate $c \in C$;
16             Optimal $c^* \leftarrow \mathsf{argmax}_{c \in C}\mathsf{imp}(\mathcal{M}, c)$;
17             Fix $t.A$ using $c^*$ and add $[t.A]_=c^*$ to $\Gamma$;
18         Remove ugly outlier $t.A$ from $U$; flag := true;
19 return Fixed dataset $\mathcal{D}_c$ with ugly outliers resolved;
```

Example 3: Continuing with Examples 1 and 2, OFix first applies rules in Σ_d to detect ugly outliers: t_1.Glucose, t_2.RecordedLabel, t_3.InsulinResistance, and t_8's HbA1c and InsulinResistance. Then, using $\mathcal{M}_{\mathsf{FixA}}$ and $\mathcal{M}_{\mathsf{FixE}}$ (trained on Diabetes and external data), OFix applies rules in Σ_f to generate candidate fixes, with conflicts: t_1.Glucose $= \{125, 113\}$, t_2.RecordedLabel $= \{0, 1\}$, t_3.InsulinResistance $= \{11.0, 6.0\}$, and t_8.HbA1c $= 6.0$, t_8.InsulinResistance $= 4.7$. Conflicts are resolved by selecting the candidate with higher $\mathsf{imp}(\mathcal{M}, c)$, yielding final repairs: t_1.Glucose $= 113$, t_2.RecordedLabel $= 1$, and t_3.InsulinResistance $= 11.0$. These updates are added to the ground truth Γ; $\mathcal{M}$ subsequently classifies t_1, t_2, t_3, and t_8 correctly on the repaired data. The process repeats until no further changes occur. □

Remark. OFix addresses chasing challenges by (1) using batch processing to infer multiple OMRs simultaneously, thus reducing time cost; and (2) resolving conflicts by objective function optimization. These ensure that each ugly outlier has a unique fix.

Complexity. OFix takes $O(|U| \times (\tau \times n + |\Sigma|) + n \times m \times w)$ time, where τ is the number of nearest neighbors in KNN, $|\Sigma|$ is the number of rules in Σ, n is the number of tuples in $\mathcal{D}$, m is the number of attributes in $\mathcal{D}$, w is the number of vertices in KB, and $|U|$ is the number of ugly outliers to be fixed (see [1] for details).

Termination and accuracy guarantees. OFix conducts deep cleaning since fixes at one chase step may help generate fixes in subsequent steps. Moreover, one can verify that wrong fixes cannot enter $\mathcal{F}_k$ if Σ and Γ are correct and ML/function predicates are accurate, by induction on chase steps. In addition, we show that OFix is Church-Rosser. A chase-based algorithm is *Church-Rosser* if for any dataset $\mathcal{D}$, set of rules Σ, and ground truth Γ, all chasing sequences of ξ by (Σ, Γ) terminate and converge at the same result, regardless of the rules used or their application order (cf. [6]).

Proposition 1: OFix *with* OMRs *is Church-Rosser.* □

Proof sketch: The proof has two steps. (1) *Any chasing sequence is finite.* Intuitively, each chasing step fixes at least one ugly outlier and the set U is finite; hence so is ξ. Specifically, for any terminal chasing sequence $\xi = (\mathcal{F}_0, \ldots, \mathcal{F}_n)$ of $\mathcal{D}$ by (Σ, Γ), we verify that $n \le |\mathcal{D}|^2 + |\Gamma||\mathcal{D}|$, since chase step (a) assigns a constant c from Γ or deduced fixes to $t.A$, which makes at most $|\Gamma||\mathcal{D}|$ steps; or (b)

Name	Domain	#Samples	#Features	%Outliers	$\#\lvert\Sigma_d\rvert$	$\#\lvert\Sigma_f\rvert$
Cardio [185]	health	2,114	21	22.04	38	89
Annthyroid [112]	health	7,200	6	7.42	21	35
Optdigits [8]	image	5,216	64	2.88	133	242
PageBlocks [119]	classification	5,393	10	9.46	35	93
Pendigits [151]	image	6,870	16	2.27	43	82
Satellite [161]	image	6,435	36	31.64	114	198
Shuttle [164]	classification	49,097	9	7.15	57	114
Yeast [122]	health	1,484	8	34.16	29	62
Census [129]	society	199,523	44	6.21	189	303
Covtype [44]	classification	581,012	55	0.47	158	269
Flights [44]	traffic	1,000,000	31	1.12	102	182

Table 2: Real-life datasets

sets two attributes equal, at most enumerating all tuple pairs ($|\mathcal{D}|^2$). The values extracted from KB via HER are bounded by $|\mathcal{D}|$.

(2) *All chasing sequences terminate at the same result.* Assume by contradiction that two terminal chasing sequences $\xi_1 = (\mathcal{F}_0, \ldots, \mathcal{F}_{n_1})$ and $\xi_2 = (\mathcal{F}'_0, \ldots, \mathcal{F}'_{n_2})$ of $\mathcal{D}$ with (Σ, Γ) have different results. Since ξ_1 and ξ_2 differ, there exists a chase step $\mathcal{F}'_i \Rightarrow_{(\varphi,h)} \mathcal{F}'_{i+1}$ in ξ_2 such that $\mathcal{F}'_{i+1}$ extends $\mathcal{F}'_i$ *w.r.t.* valuation $h(t').A$ of a tuple variable t' of φ but $h(t').A$ is not in $\mathcal{F}_{n_1}$ of ξ_1. However, we verify that $\mathcal{F}_{n_1} \Rightarrow_{(\varphi,h)} \mathcal{F}_{n_1+1}$ is a valid chase step expanding $\mathcal{F}_{n_1}$ *w.r.t.* $h(t').A$ by induction on the length of ξ_1, contradicting the assumption that ξ_1 is terminal. Once training completes, the predictions of ML models in OMRs are consistent across ξ_1 and ξ_2 (see [1]). □

Novelty of OFix. Unlike existing imputation methods, OFix (1) combines logical reasoning with ML prediction to reduce false positives/negatives of ML-based imputation (one of our strategies in Section 5.1), as shown by prior work [50] and our ablation study (0.5% accuracy improvement, Section 6); (2) extends the chase to recursively refine later outliers using earlier corrections; and (3) targets ugly outliers, unlike heuristics that fix all missing values. Different from iterative optimization in ML, the chase recursively propagates fixes in a deterministic process and converges to a unique result (Church-Rosser [6]). Moreover, the chase provides binary accuracy guarantee that determines whether the repaired data meets integrity requirements within the logical framework of data cleaning.

6 EXPERIMENTAL STUDY

Using real-life and synthetic datasets, we empirically verified the accuracy, robustness and scalability of OHunt for detecting and fixing ugly outliers, and its impact on the accuracy of ML classifiers.

Experimental setting. We start with the setting.

Datasets. We used 18 real-life datasets with different sizes and outlier rates from different domains (*e.g.,* health); 11 are in Table 2 (see [1] for the other 7). Pendigits, PageBlocks, Optdigits, Cardio, Shuttle, Yeast, Satellite, Census and Annthyroid are from outlier detection benchmarks, and the rest come from public repositories.

Besides testing real outliers, we created 20 synthetic datasets based on real data to test the robustness of OHunt. Each dataset is injected with one of four types of noises: local, global, cluster and dependency [73]. Local noise refers to outliers deviating from their local neighborhoods [27, 73]. Global noise exhibits larger deviations from normal data [73, 85], generated from a uniform distribution. Cluster noises [104] are groups of outlier instances that share similar characteristics and appear close together in the feature space [49, 109]. Dependency noise involves deviations from the typical dependency structure of normal data [73, 120].

We split each dataset into 70% training, 20% validation, and 10% testing. The training set was used to learn models and OMR rules; the validation set guided hyperparameter tuning and cleaning

$\mathcal{M}$	$\mathcal{D}$	ABOD	ECOD	Iforest	RePEN	SLAD	ICL	NeuTraL	uOMRs
SVM	Annthyroid	0.000	0.237	0.218	0.212	0.090	0.292	0.267	**0.932**
	Average	0.195	0.335	0.327	0.233	0.579	0.163	0.300	**0.919**
RF	Yeast	0.163	0.195	0.182	0.167	0.160	0.231	0.263	**1.000**
	Average	0.424	0.383	0.379	0.433	0.383	0.425	0.443	**0.926**
Softmax	PageBlocks	0.360	0.349	0.333	0.322	0.016	0.364	0.250	**0.940**
	Average	0.450	0.467	0.465	0.567	0.209	0.446	0.342	**0.915**
TabPFN	Shuttle	0.560	0.483	0.385	0.342	0.412	0.581	0.759	**0.895**
	Average	0.564	0.491	0.490	0.467	0.250	0.424	0.564	**0.931**

Table 3: F1-score for Unsupervised Ugly Outlier Detection

$\mathcal{M}$	$\mathcal{D}$	CatB	LGB	XGB	DevNet	DSAD	RoSAS	PReNeT	sOMRs
SVM	Annthyroid	0.051	0.089	0.063	0.298	0.175	0.544	0.281	**0.987**
	Average	0.057	0.041	0.038	0.339	0.297	0.532	0.502	**0.953**
RF	Yeast	0.088	0.084	0.103	0.123	0.162	0.178	0.247	**0.994**
	Average	0.049	0.062	0.107	0.258	0.289	0.429	0.447	**0.946**
Softmax	PageBlocks	0.068	0.135	0.137	0.570	0.410	0.496	0.624	**1.000**
	Average	0.070	0.110	0.115	0.570	0.499	0.664	0.776	**0.962**
TabPFN	Shuttle	0.000	0.000	0.000	0.533	0.378	0.154	0.148	**1.000**
	Average	0.077	0.085	0.082	0.322	0.320	0.177	0.249	**0.968**

Table 4: F1-score for Label-based Ugly Outlier Detection

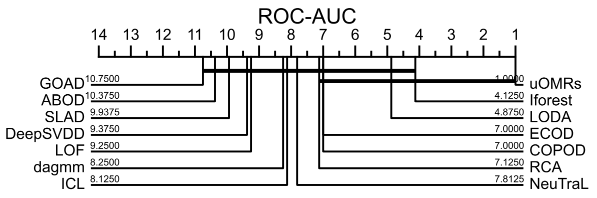
strategy selection via Bayesian optimization; and the test set was reserved only for final evaluation of accuracy and robustness.

Ground truth of good/bad/ugly *outliers*. (a) All datasets include ground-truth outlier labels annotated by domain experts and widely adopted in prior ML studies [43, 108**?**]. (b) Ugly outliers are identified through experiments as those causing misclassifications or violating domain knowledge. (c) Good outliers are rare but valid instances aligned with domain rules, and are useful for learning long-tail patterns. (d) All remaining ones are labeled as bad outliers.

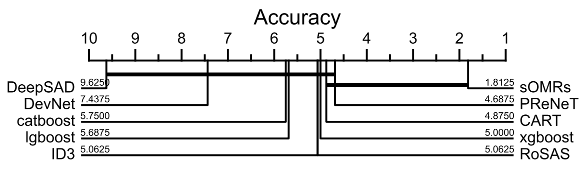
ML classifiers. We tested 7 ML classifiers $\mathcal{M}$: (1) Support Vector Machine (SVM) [162], for high-dimensional spaces via maximum-margin decision boundaries; (2) Naive Bayes (NB) [19], by Bayesian inference with feature independence; (3) Random Forest (RF) [26], an ensemble of decision trees via voting; (4) Softmax [28], widely used in neural networks for multi-class classification; (5) Decision Tree (DT) [139], a hierarchical, interpretable model based on feature splits; (6) Tabpfn [83], a transformer-based deep learning method for small tabular data; (7) XGBoost [34], a fast and accurate gradient boosting algorithm. Unless stated otherwise, XGBoost is the default.

Baselines. We compared OHunt with 22 outlier detection baselines following [42, 73, 181, 182], integrated with outlier correction methods in Exp-2. (1) Unsupervised ones: (a) Statistical: ECOD [108], COPOD [107], LODA [134] and RCA [110]. (b) Distance-based: ABOD [100]. (c) Ensemble-based: IForest [112]. (d) Deep leaning: DAGMM [192], SLAD [184], GOAD [21], DSVDD [147], REPEN [128], ICL [157] and NeuTraL [138]. (2) Semi-supervised: PReNet [130], DevNet [131], DSAD [148] and ROSAS [183]. (3) Supervised: CatBoost [137], LGBoost [91] and XGBoost [34]. (4) Rule-based ones include ID3 [42, 139] and CART [42, 116].

Rules. We learned OMRs from 18 real-world datasets. The number of OMRs for detecting ($|\Sigma_d|$) and fixing outliers ($|\Sigma_f|$) are shown in Table 2. Based on the availability of labels during training, we classify OMRs learned into uOMRs and sOMRs. We compared uOMRs with unsupervised baselines, and sOMRs with label-based (supervised, semi-supervised, and rule-based) baselines. The average sizes of $|\Sigma_d|$ and $|\Sigma_f|$ across all datasets are 67 and 134, respectively.



(a) CD plot of ROC-AUC on unsupervised algorithms



(b) CD plot of Accuracy on label-based algorithms

Figure 1: Performance Comparison of OMRs and Baselines in Binary Outlier Detection Tasks

Accuracy metrics. We adopt the following: (1) Accuracy defined in Section 2.1. (2) Recall (R), the ratio of actual outliers correctly predicted as positive. (3) F1-score = $2 \times \frac{P \times R}{P+R}$, where P is the ratio of actual outliers to all predicted positives. (4) Average Precision (AP) [171], the weighted average of P-values across different recall levels, typically used for imbalanced datasets. (5) ROC-AUC [74] is used for binary classification, where a value closer to 1 indicates better performance. (6) Critical Difference (CD) Diagram [40, 89] visualize and compare the performance of multiple models.

Environmental setting. We conducted all experiments on a server equipped with an Intel Core 2.90 GHz CPU and 32 GB of memory. OHunt and all the baselines were implemented in Python 3.8. Each experiment was run 3 times, and the average result is reported here.

Experimental findings. We next report our findings.

Exp-1: Outlier detection: Accuracy and robustness. We tested OHunt for: (1) Proportions of good, bad, and ugly outliers in real-life datasets and F1-score for ugly outliers. (2) Robustness in detecting ugly outliers in synthetic datasets with varying noise ratios and types. (3) Effectiveness in distinguishing outliers in binary classification datasets [24] using CD diagrams.

(1) Accuracy. We start with the detection of different outliers.

(a) The good, the bad, and the ugly. (i) Across all real-world datasets tested, outliers range from 0.47% to 49.90%, 14.08% on average. On average, good outliers make up 80.33% of all outliers, *e.g.,* 69.44% in the Apple dataset and 95.25% in Covtype. Bad outliers are 19.67% on average, in which the ratio of ugly outliers varies significantly. For instance, in Star, 97.06% of the 1,103 bad outliers are ugly, versus an average of 64.72% across all datasets. (ii) Predicates $\mathcal{M}_{\text{DetO}}(\cdot)$ and outlier$(\cdot)$ are effective in identifying outliers, while loss$(\cdot)$ distinguishes good and bad ones. During the training of $\mathcal{M}$, loss$(\cdot)$ for good outliers decreases below a threshold λ learned via Bayesian optimization, while it remains stable for bad outliers. (iii) Model $\mathcal{M}_{\text{DetI}}(\cdot)$ identifies influential features for ugly outliers, while function imbalanced$(\cdot)$ picks out imbalanced ugly outliers in bad ones.

(b) F1-score of ugly outliers. We evaluated the F1-score of ugly outliers from 18 real datasets using four classifiers (SVM, RF, Softmax and TabPFN). We compared the accuracy of uOMRs (resp. sOMRs) against unsupervised (resp. label-based) baselines. We report the average and a couple of individual cases. As shown in Tables 3 and 4, on average, (i) The average F1-score of uOMRs is 0.923 (up to 0.997), 55.3% higher than unsupervised baselines. (ii) For sOMRs, the average F1-score is 0.957 (up to 1.000), 68.5% higher than label-based baselines. These results verify that OHunt is more effective in detecting outliers that negatively impact ML classifiers.

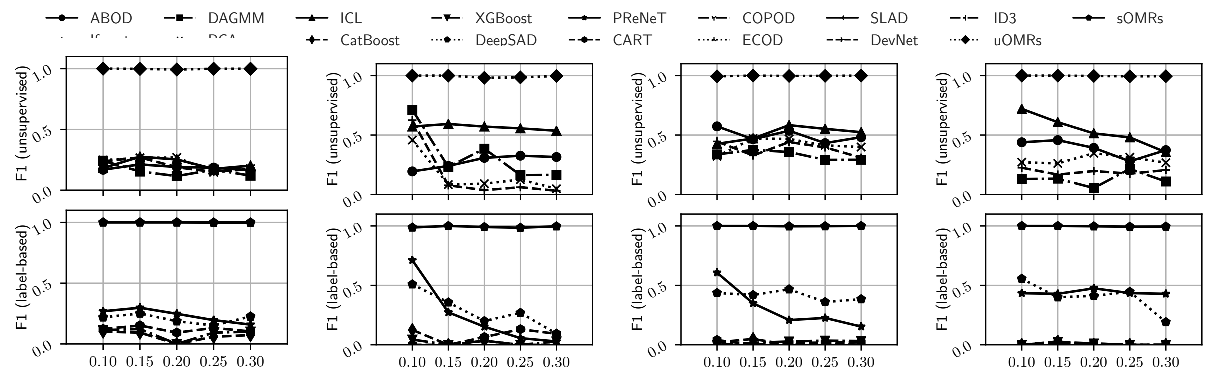
(2) Robustness. Besides real outliers, we injected one of local, global, cluster and dependency noise into real datasets Cardio, PageBlocks (PB), Annthyroid (ATR) and Waveform (WF), by varying its ratio at 0.10/0.15/0.20/0.25/0.30, yielding 20 noisy datasets. Here k_l, k_g, k_c and k_d denote the ratio of the four types of noise, respectively.

As shown in Figure 2(a-d), (a) OHunt is robust to all four types of noise, whereas the baselines are most sensitive to global noise. In PageBlocks, as k_g increases from 0.10 to 0.30, the F1-score for detecting ugly outliers with uOMRs decreases by 0.005 (compared to the unsupervised baselines' average reduction of 0.101), while the F1-score of sOMRs decreases by 0.005 (compared to the average reduction of 0.074 in the label-based baselines). (b) For high noise ratios, OHunt consistently outperforms the baselines; its average F1-score is 81.4% higher, up to 100.0%. Even in the noise-sensitive Waveform dataset (k_d is 0.3), the F1-score of OHunt is consistently above 0.995, while it is below 0.430 for the baselines; this demonstrates OHunt's robustness against noise. (c) For each noise type, when ratio k_l, k_g, k_c or k_d increases, the F1-score for detecting ugly outliers decreases for all baselines. For instance, in Cardio with local noise, the average F1-score of baselines declines from 0.190 to 0.150 as k_l increases from 0.1 to 0.3. Similarly, on Annthyroid with cluster noise, it drops from 0.321 to 0.252 as k_c increases over the same range. This said, OHunt still outperforms all the baselines.

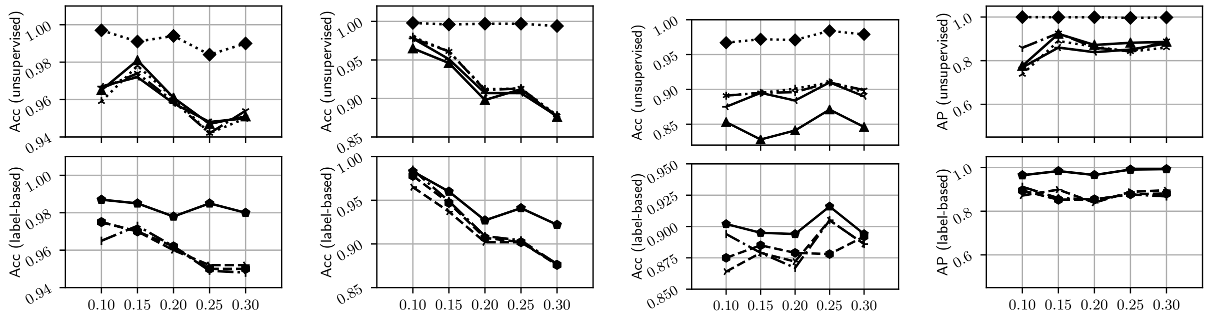
(3) Classical tasks. We evaluated the performance of OHunt versus 22 baselines on 18 real-world datasets for classical outlier detection, *i.e.,* distinguishing outliers from binary datasets composed of normal and outlier data. The analysis of Accuracy and ROC-AUC metrics is based on CD diagrams. As shown by examples in Figure 1(a-b), uOMRs beat 13 unsupervised methods in terms of ROC-AUC. For Accuracy, sOMRs ranks the first among 10 label-based methods. These verify that OHunt is effective in classical outlier detection.

Exp-2: Fixing ugly outliers: Accuracy and robustness. We evaluated OHunt's effectiveness in fixing outliers and improving ML classifier accuracy, focusing on: (1) the impact on the F1-score of $\mathcal{M}$, (2) the impact of fixing outliers vs. removing tuples, and (3) OHunt's robustness to varying outlier ratios (0.10-0.30) and types (local/global/cluster/dependency). Since baselines either detect or correct outliers, we combine detectors with correction methods (see Section 7) for a fair comparison with OHunt. Specifically, we combine (1) elimination and statistics-based correction methods (*e.g.,* [88]) with unsupervised detectors (*e.g.,* SLAD); (2) ML-based (*e.g.,* [170]), CleanML [106] and DB4AI methods (*e.g.,* [59]) for label-based correction, with semi-supervised, supervised, and rule-based detectors (*e.g.,* CART); and (3) for uOMRs and sOMRs in OHunt, we use the chasing process in the OFix algorithm for outlier correction.

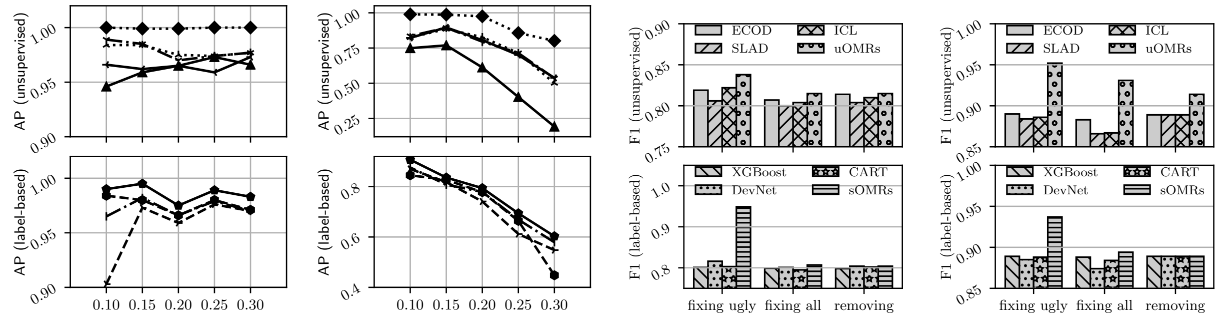
(1) Accuracy. We used 18 multi-class datasets (e.g., DryBean) and real outlier datasets (e.g., Shuttle). We trained seven classifiers $\mathcal{M}$ (SVM, Softmax, DT, RF, NB, XGBoost, and TabPFN) on the training data, tuned hyperparameters on the validation set, and evaluated F1-scores on the test set. OHunt and baselines were then used to detect and repair ugly outliers in $\mathcal{D}$. Finally, $\mathcal{M}$ was retrained on



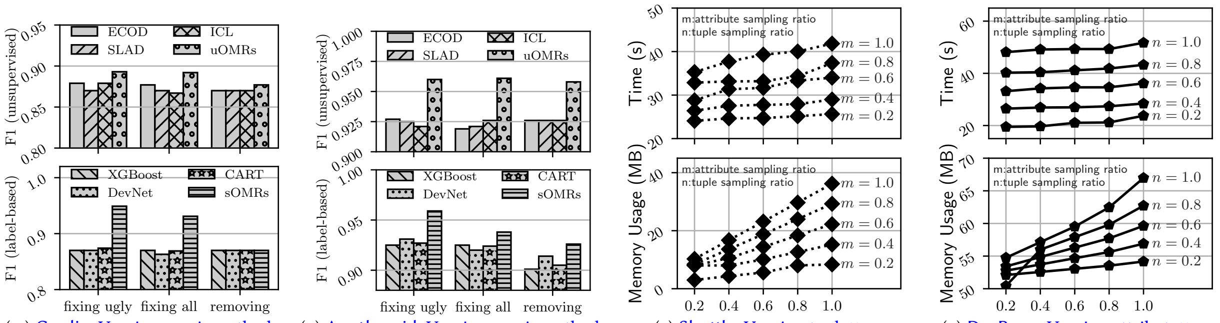
(a) Cardio: F1 of detecting varying k_l (b) PB: F1 of detecting varying k_g (c) ATR: F1 of detecting varying k_c (d) WF: F1 of detecting varying k_d



(e) ATR: Acc of fixing varying k_c (f) PB: Acc of fixing varying k_a (g) WF: Acc of fixing varying k_d (h) ATR: AP of fixing varying k_c



(i) PB: AP of fixing varying k_g (j) WF: AP of fixing varying k_d (k) Census: Varying repair methods (l) Star: Varying repair methods



(m) Cardio: Varying repair methods (n) Annthyroid: Varying repair methods (o) Shuttle: Varying tuple% n (p) DryBean: Varying attribute% m

Figure 2: Performance evaluation of OHunt **(u**OMRs**/s**OMRs**) and baselines**

the cleaned training data and re-evaluated on the cleaned test set.

As shown in Table 5, OHunt improves $\mathcal{M}$'s F1-score by an average of 7.2%, up to 34.8%, outperforming the baselines by an average of 7.2% and up to 25.2%. More specifically, uOMRs outperform the baselines by 6.3% on average, up to 25.2%, and sOMRs improve by 8.1%, up to 34.8%. For instance, the F1-score of TabPFN with sOMRs is 0.944 on Yeast, compared to 0.692 with CART for detection and CleanML for repair; this makes a 25.2% improvement. We find that detecting and fixing ugly outliers is more effective than repairing all outliers in baselines. This is because (a) the latter may incorrectly repair good and bad outliers that should not be fixed, introducing new errors and altering the normal data distribution of $\mathcal{D}$; and (b) the latter cannot guarantee the Church-Rosser property when fixing ugly outliers, degrading the data quality in the test set.

$\mathcal{M}$	$\mathcal{D}$	F1($\mathcal{M}, \mathcal{D}$)	ABOD	ECOD	LODA	DSVDD	RePEN	SLAD	ICL	NeuTraL	**u**OMRs	DevNet	DSAD	RoSAS	PReNet	ID3	CART	**s**OMRs
SVM	DryBean	0.860	0.868	0.873	0.859	0.873	0.874	0.882	0.869	0.882	**0.949**	0.856	0.877	0.879	0.867	0.873	0.866	0.955
	Online	0.824	0.817	0.821	0.820	0.817	0.823	0.808	0.816	0.805	**0.875**	0.804	0.808	0.800	0.797	0.817	0.820	0.846
	Average	0.875	0.878	0.877	0.876	0.878	0.881	0.879	0.880	0.881	**0.917**	0.874	0.878	0.879	0.878	0.878	0.878	0.933
RF	Census	0.796	0.804	0.798	0.800	0.801	0.797	0.807	0.805	0.809	**0.845**	0.800	0.808	0.805	0.817	0.806	0.789	0.970
	Star	0.903	0.891	0.893	0.881	0.881	0.886	0.877	0.882	0.886	**0.945**	0.874	0.880	0.883	0.874	0.885	0.884	0.973
	Average	0.826	0.825	0.826	0.826	0.823	0.826	0.827	0.829	0.826	**0.883**	0.823	0.828	0.817	0.824	0.831	0.824	0.932
Softmax	Satellite	0.853	0.845	0.846	0.843	0.842	0.846	0.833	0.842	0.844	**0.862**	0.846	0.845	0.846	0.847	0.841	0.848	0.965
	Student	0.631	0.656	0.667	0.681	0.686	0.667	0.651	0.664	0.659	**0.795**	0.650	0.667	0.642	0.661	0.657	0.658	0.729
	Average	0.779	0.787	0.793	0.794	0.788	0.789	0.784	0.791	0.789	**0.856**	0.784	0.788	0.777	0.780	0.786	0.790	0.860
DT	Yeast	0.651	0.666	0.655	0.660	0.629	0.647	0.672	0.676	0.649	**0.727**	0.645	0.645	0.642	0.660	0.651	0.664	0.842
	Cardio	0.858	0.842	0.848	0.859	0.848	0.863	0.852	0.869	0.869	**0.876**	0.852	0.853	0.863	0.863	0.863	0.863	0.953
	Average	0.801	0.803	0.803	0.803	0.790	0.802	0.798	0.801	0.793	**0.852**	0.788	0.799	0.787	0.790	0.801	0.802	0.875
NB	Online	0.791	0.790	0.793	0.793	0.786	0.795	0.799	0.797	0.797	**0.810**	0.784	0.782	0.769	0.770	0.793	0.790	0.825
	Shuttle	0.951	0.952	0.951	0.952	0.952	0.951	0.986	0.985	0.986	**0.990**	0.952	0.951	0.950	0.951	0.951	0.951	0.952
	Average	0.816	0.820	0.820	0.819	0.819	0.822	0.820	0.830	0.826	**0.859**	0.812	0.818	0.818	0.805	0.817	0.818	0.880
XGBoost	DryBean	0.865	0.870	0.871	0.848	0.856	0.853	0.857	0.868	0.847	**0.962**	0.836	0.863	0.840	0.856	0.864	0.849	0.947
	Census	0.809	0.809	0.812	0.813	0.813	0.812	0.814	0.811	0.817	**0.847**	0.807	0.809	0.803	0.815	0.810	0.810	0.971
	Average	0.876	0.879	0.879	0.872	0.872	0.874	0.875	0.876	0.876	**0.935**	0.867	0.876	0.869	0.873	0.876	0.872	0.958
TabPFN	PageBlocks	0.956	0.910	0.883	0.927	0.883	0.905	0.942	0.942	0.925	**0.979**	0.908	0.934	0.904	0.867	0.938	0.925	0.989
	Yeast	0.596	0.739	0.720	0.696	0.716	0.744	0.716	0.716	0.716	**0.848**	0.716	0.692	0.734	0.755	0.692	0.692	0.944
	Average	0.813	0.825	0.700	0.820	0.812	0.814	0.822	0.802	0.796	**0.922**	0.809	0.809	0.828	0.819	0.822	0.799	0.917

Table 5: Comparison of OMRs and Outlier Detection Baselines in Downstream Classification F1-Score

(2) Robustness. In the same setting as the robustness tests in Exp-1, we report acc($\mathcal{M}, \mathcal{D}_c$) and AP scores of $\mathcal{M}$ under different noise ratios and types. Here acc($\mathcal{M}, \mathcal{D}_c$) measures the improvement in classification accuracy by OHunt compared to baselines, while AP highlights OHunt's robustness on imbalanced datasets.

As shown in Figure 2(e-j), (a) for different noise ratios and types, OHunt outperforms all the baselines in acc($\mathcal{M}, \mathcal{D}_c$) and AP by 3.8% and 6.6% on average, respectively, up to 13.3% and 60.9%. OHunt is less sensitive to noise than all the baselines. This is because it employs functions for influential features and loss when detecting ugly outliers, and logical reasoning to reduce false positives/negatives of ML-based detectors. When fixing ugly outliers, OHunt combines logical deduction, external data and statistical methods, rather than removing these outliers, which is crucial when noise ratio is high. (b) While sOMRs are more sensitive to global noise, both sOMRs and uOMRs are sensitive to dependency noise. For example, on the PageBlocks dataset, when the global noise ratio increases from 0.1 to 0.3 , the accuracy score of sOMRs drops from 0.983 to 0.922. Similarly, on the Waveform dataset, when the dependency noise ratio rises from 0.1 to 0.3, the AP score of uOMRs (resp. sOMRs) decreases from 0.989 (resp. 0.907) to 0.801 (resp. 0.603).

(3) Ablation study. We evaluated the impact of (a) various detectors, (b) correction strategies: fixing only ugly outliers vs. all outliers, and removing ugly tuples vs. fixing ugly outliers in place, and (c) the impact of logical reasoning, ML predicates and function predicates.

(a-b) We tested eight outlier detectors and three cleaning strategies using four real datasets: Census, Star, Cardio and Annthyroid. We investigated the impacts of removing ugly tuples (OHunt$_{\text{rem}}$), fixing ugly outliers (OHunt$_{\text{ugly}}$), and fixing all outliers (OHunt$_{\text{all}}$) on the F1-score of the XGBoost classifier ($\mathcal{M}$). The selected detectors include OMRs (uOMRs and sOMRs), three unsupervised baselines (*e.g.,* SLAD), and three label-based baselines (*e.g.,* rule-based CART).

As shown in Figure 2(k-n), (i) by using uOMRs and sOMRs, OHunt$_{\text{rem}}$, OHunt$_{\text{ugly}}$ and OHunt$_{\text{all}}$ improve the F1-scores of $\mathcal{M}$ better than with all unsupervised and label-based baselines. For example, for OHunt$_{\text{ugly}}$ on Cardio, the F1-score of uOMRs and sOMRs is 0.893 and 0.949, respectively, while it is 0.879 and 0.874 for the best unsupervised (ECOD) and label-based (CART) baselines. This suggests that OHunt is able to accurately detect and fix ugly outliers responsible for classification errors. (ii) Under the F1 metric, OHunt$_{\text{ugly}}$ with uOMRs outperforms OHunt$_{\text{all}}$ with uOMRs in improving the F1-score of $\mathcal{M}$, achieving a slightly higher average score (0.911 vs. 0.900). Both methods perform better than OHunt$_{\text{rem}}$ (0.891 on average). (iii) For sOMRs, OHunt$_{\text{ugly}}$ has a greater impact on the F1-score of $\mathcal{M}$ than OHunt$_{\text{all}}$, with average F1-score of 0.949 and 0.893, respectively. Both OHunt$_{\text{ugly}}$ and OHunt$_{\text{all}}$ consistently beat OHunt$_{\text{rem}}$, with which the average F1-score of $\mathcal{M}$ is 0.872.

(c) We evaluated the impact of OMR predicates by removing fixing predicates (OHunt$_{\text{nofix}}$), ML predicates (OHunt$_{\text{noml}}$), logical reasoning (OHunt$_{\text{nolr}}$), or function predicates (OHunt$_{\text{nofun}}$). We fix ugly outliers and use uOMRs and sOMRs as detectors. The datasets, metrics and classifier are the same as in (a-b). We find that OHunt$_{\text{nofix}}$ and OHunt$_{\text{noml}}$ do not improve $\mathcal{M}$'s F1-score, since they are unable to conduct the chase in OFix or learn high-quality rules in OLeaner. Compared to the complete OHunt, (i) the F1-score of classifier $\mathcal{M}$ is 0.3% lower for uOMRs and 0.6% lower for sOMRs with OHunt$_{\text{nolr}}$ since it cannot reduce false positives/negatives of ML predictions; and (ii) it is 1.7% lower for uOMRs and 6.4% for sOMRs with OHunt$_{\text{nofun}}$ due to the inability to distinguish good, bad, and ugly outliers without function predicate loss($\mathcal{M}, \mathcal{D}, t, A$). The ablation test on OHunt$_{\text{nolr}}$ shows that logical reasoning plays a key role in fixing ugly outliers, where logic predicates can reduce its false positives and negatives, as verified by prior work [51] and our ablation study (0.6% more accurate than OHunt$_{\text{nolr}}$).

Exp-3: Scalability. We also studied the scalability of OHunt using real datasets, by varying (1) the sampling ratio n of tuples; (2) the sampling ratio m of attributes. We also evaluated (3) the scalability of rule discovery with $|\mathcal{D}|$, and Bayesian parameters $|\Theta|$ and $|T|$.

(1) Varying number of tuples in $\mathcal{D}$. Fixing the attribute sampling ratio m at 0.2, 0.4, 0.6, 0.8 and 1.0, we varied the tuple ratio n across 0.2, 0.4, 0.6, 0.8 and 1.0 via random tuple sampling, and tested the runtime and maximum memory usage of OHunt on the Shuttle dataset using the RF classifier. As shown in Figure 2(o), (a) the runtime and memory usage of uOMRs and sOMRs increase nearly linearly with

n. (b) For both outlier detection and correction, sOMRs take longer and consume more memory than uOMRs. For instance, when both row and column sampling ratios are set to 1.0, uOMRs take 41.829s and use 36.210 MB, while sOMRs take 88.173s and use 66.27 MB.

(2) Varying number of attributes in $\mathcal{D}$. Fixing the tuple sampling ratio n at 0.2, 0.4, 0.6, 0.8 and 1.0, we varied the attribute ratio m across 0.2, 0.4, 0.6, 0.8 and 1.0 on the DryBean dataset using XGBoost. As shown in Figure 2(p), (a) the runtime and memory usage of OHunt increase almost linearly with m; and (b) sOMRs use more time and memory than uOMRs for better accuracy, with $\text{acc}(\mathcal{M}, \mathcal{D}_c) = 0.958$ for sOMRs and $\text{acc}(\mathcal{M}, \mathcal{D}_c) = 0.939$ for uOMRs.

(3) Rule discovery. We tested the scalability of OMR discovery algorithm OLeaner, by varying (a) the average size $|\Theta|$ of configurations Θ, (b) the total number $|T|$ of tasks T, and (c) the data size $|\mathcal{D}|$ on Flights. (a) Fixing $|T|$ at 10 and $|\mathcal{D}|$ at 20,000, while varying $|\Theta|$ at 3, 4, 5, 6 and 7, the runtime of OLeaner ranges over 134.35s, 129.26s, 133.02s, 130.79s and 135.44s. This shows that OLeaner is insensitive to $|\Theta|$, since it always searches for the optimal configuration within Θ. (b) Fixing $|\Theta|$ at 5 and $|\mathcal{D}|$ at 20,000, while varying $|T|$ at 10, 15, 20, 25 and 30, OLeaner takes 140.54s, 205.86s, 277.21s, 338.17s and 384.83s, respectively. This shows that OLeaner scales nearly linearly with $|T|$, since it sequentially executes tasks in T and leverages information from previous ones. (c) Fixing $|\Theta|$ at 5 and $|T|$ at 5, while varying $|\mathcal{D}|$ across 10,000, 20,000, 30,000, 40,000 and 50,000, OLeaner takes 13.32s, 66.77s, 170.25s, 300.04s and 460.63s, and finds 86, 160, 77, 85 and 148 useful OMRs, respectively. The runtime of OLeaner grows approximately linearly with $|\mathcal{D}|$. OLeaner scales well with $|\Theta|$, $|T|$ and $|\mathcal{D}|$ because the precise objective function in Bayesian optimization, which reduces irrelevant predicate selection and facilitates rapid convergence to the optimal hyperparameters.

We compared OLeaner with two SOTA levelwise baselines, DCFinder [133] and PRMiner [56], which discover DCs [12] and REEs [57], but cannot determine hyperparameters or mine OMRs. We find that the baselines mined very few helpful rules when varying $|\mathcal{D}|$ across 10,000, 20,000, 30,000, 40,000 and 50,000 on Flights. (1) DCFinder found 9,509 rules with support $0.00002|\mathcal{D}|$ and confidence 0.95, taking from 4,258s to 8,463s on five datasets. (2) PRMiner mined 991 REEs with the same support and confidence, running from 385.24s to 1941.77s. However, most rules were logical reasoning rules with limited utility. All helpful REEs and DCs were also found by OLeaner, contributing at most 0.6% accuracy improvement to $\mathcal{M}$. This highlights the need for a meta-learning approach to efficiently discover OMRs and improve $\mathcal{M}$'s accuracy.

Exp-4: Case study. To evaluate OMRs' interpretability on ugly outliers, we conducted experiments on two large datasets Flights and Census, using XGBoost and SVM as classifiers to predict outliers. We applied OMR rules to explain these outliers and compared with LIME [143] and SHAP [117] as baselines. Interpretability was assessed using fidelity [143] (whether the prediction remains consistent after retaining only selected features), and sparsity [143] (the ratio of features used in the explanation to total features).

We provide explanations for 100 randomly selected test samples predicted as outliers by $\mathcal{M}$, and evaluate the explanations in terms of fidelity and sparsity. (1) On the Flights dataset, OMRs achieved an average sparsity of 0.140 and perfect fidelity (1.000) on XGBoost. In comparison, LIME had a sparsity of 0.333 and fidelity of 1.000, while SHAP showed 0.293 sparsity and 1.000 fidelity. For SVM, OMRs achieved 0.152 sparsity and 1.000 fidelity, compared to LIME's 0.333 sparsity and 0.980 fidelity, and SHAP's 0.278 sparsity and 0.970 fidelity. (2) On the Census dataset, OMRs achieved sparsity of 0.170 and fidelity of 0.950 for XGBoost, and 0.125 and 1.000 for SVM. In comparison, LIME showed 0.832 sparsity and 0.900 fidelity for both classifiers, while SHAP achieved 0.580 sparsity and 0.920 fidelity for XGBoost, and 0.422 sparsity and 0.800 fidelity for SVM. Compared to LIME and SHAP, OMRs maintain higher fidelity but use fewer key features, providing stronger interpretability.

To further assess the interpretability, we selected ugly outliers from Diabetes causing $\mathcal{M}$'s misclassifications. We compared with CAPE and ExTuNe as baselines. Outliers in Table 1 are explained using OMRsfor Example 1 and in Section 2.2. We find the following.

(1) CAPE requires a predefined schema and manual outlier specification, making it impractical for flat tables without defined schemas. For comparison, we manually built a schema and marked four tuples (t_1, t_2, t_3, t_8) for CAPE. CAPE explained t_1 using a reverse outlier pattern, but it failed to explain t_2 (mislabeled), t_3 (rare good outlier), and t_8 (combination-based outlier). Overall, CAPE does not effectively explain the outliers detected by OMRs.

(2) We also used ExTuNe to identify and explain the top-4 inconsistent tuples, highlighting key features such as "Glucose", "HbA1c", and "InsulinResistance". However, ExTuNe incorrectly ranked t_5, t_6, t_2, and t_1 as the top outliers, misclassifying t_5 and t_6 as ugly outliers and t_2 as a feature outlier instead of a label error. It also struggled to detect label and combination outliers.

In contrast to CAPE and ExTuNe, OMRs are able to accurately detect good, bad, and ugly outliers, providing faithful explanations tied to key features, labels, and the classifier $\mathcal{M}$.

Summary. From the experiments we find the following.

(1) Although ugly outliers are rare in real datasets, OHunt detects them much better than the baselines. On average, 14.08% of samples are outliers, with 80.33% being good and 19.67% bad, of which 64.72% are ugly. OHunt achieves high F1-scores for detecting ugly outliers, 0.923 with uOMRs and 0.957 with sOMRs, outperforming the baselines by an average of 61.9%, up to 68.5%.

Ugly outliers are rare in datasets because most errors do not significantly affect the classifier's decision, and only those near the decision boundary are likely to cause misclassification [38, 98, 154]. OMRs effectively identify such cases by considering the impact of outliers on the classifier's loss function, and by integrating function/ML predicates with logic reasoning to distinguish ugly outliers.

(2) Fixing the ugly outliers with OMRs improves the average F1-scores of $\mathcal{M}$ by 7.2%, up to 34.8%, 7.2% better than the baselines. This is attributed to OMRs' ability to effectively detect ugly outliers and apply an extended chase process that selects optimal fixes for ugly outliers using multiple repair strategies.

(3) OHunt is robust to local, global, dependency and cluster noise. Its F1-score for ugly outliers is 0.995 even with 30% dependency noise. After fixing the ugly outliers, it improves $\mathcal{M}$'s accuracy by 3.8% on average, up to 13.3% across 15 noisy datasets. This is because these types of noise can be effectively captured by predicates $\mathcal{M}_{\text{DetO}}(\cdot)$,

outlier(·), imbalanced(·) and loss(·), thus enabling accurate repair.

(4) Combining logical reasoning with ML/function predicates boosts the classification performance of $\mathcal{M}$, improving $\mathcal{M}$'s F1-score by an average of 0.5% and 4.1% compared to using either alone. This is because logic reasoning can effectively reduce the FPs/FNs of $\mathcal{M}$ [51].

(5) OHunt scales well for outlier detection and fixing. Its runtime and memory usage grow nearly linearly with the number of tuples and attributes. This efficiency stems from the extened chase.

(6) OLeaner scales near-linearly with $|\mathcal{D}|$ and $|T|$, and is insensitive to $|\Theta|$, due to its Bayesian-based optimization algorithm.

(7) While sOMRs outperforms label-based baselines, it shows no clear advantage over unsupervised uOMRs on some datasets due to (a) overfitting from limited outlier labels in small datasets, and (b) uOMRs leveraging supervisory signals from OLeaner and OFix, achieving greater speed and robustness without labeled guidance.

7 RELATED WORK

We categorize the related work as follows.

Outlier detection. Such methods can be classified as follows [32, 41, 78, 78, 150]. (1) Unsupervised methods heuristically score outliers based on their deviation; *e.g.,* [153] separates outliers by learning a decision boundary; [112] progressively isolates outliers with random trees; [150] measures the similarity between data points and their neighbors; [78] adopts clustering; [107] (resp. [108]) uses copula models (resp. CDFs) to detect outliers; and [31] suggests data-driven detector selection. (2) Semi-supervised methods train neural network models [128, 131, 189] on partially labeled datasets [73]; *e.g.,* [128] adopts random distance-based detectors; [131] computes statistical deviation scores; [189] integrates DAGMM's architecture [192] with DevNet's deviation loss; [188] enhances feature representations with unsupervised detectors and combines them with original features for classification. (3) Supervised methods train outlier detectors [19, 26, 32, 38, 70, 144] using fully labeled data [73].(4) Rule-based methods include decision trees [18, 139], iterative rule generation [11, 37], if-then-elseif rules [173], and Explanation Table [47]. These detectors are difficult to integrate with ML models, since their classification functions are highly customized, limiting their applicability to different outlier types [42].

This work departs from prior methods by studying the impact of outliers on downstream tasks. (a) We categorize outliers as good, bad, or ugly and handle them differently. (b) We combine logic reasoning and ML predictions by embedding ML detectors and loss/statistical functions as OMR predicates to improve detection accuracy. (c) We reduce FPs and FNs and explain ML predictions.

Outlier correction. Such methods are classified as follows. (1) Elimination methods directly remove the detected outliers [27, 50, 96, 112, 153]. This approach is effective when outliers have a low rate and are not representative [9]. (2) Statistics-based methods estimate replacement values for outliers [9], such as percentile-based estimation [88], deterministic-function-based transformation [167], and Bayesian statistics [67]. (3) ML-based methods, *e.g.,* [46] learns the joint distribution of clean data and fixes outliers by maximizing a posteriori [22] inference via generative models, [170] reconstructs the original data and replaces outliers accordingly, and [175] builds an uncertainty model to estimate replacement of outliers.

There have also been work on data cleaning for AI [2, 58, 59, 71, 101–103, 115], by error detection, correction and data augmentation. CleanML [106] tests various ML models on noisy datasets with common errors, and data cleaning methods for each model, which systematically investigates impacts of data quality on the models. ActiveClean [102], CoCo [58] and CHEF [179] improve data quality and model accuracy via iterative cleaning and training. HOD [32] simplifies data cleaning by asking few user-friendly questions.

In contrast, (a) we focus solely on detecting and fixing ugly outliers to improve ML accuracy, unlike prior work [2, 58, 59, 71, 101–103, 115] that target broader error types. (b) We combine logic reasoning and ML models for "deep cleaning" of ugly outliers, ensuring Church-Rosser consistency and deriving fixes as logical consequences of rules, ground truth, and predictions. (c) We use external data sources to improve fix accuracy. (d) For imbalanced data, we recommend ROC-AUC and AP to evaluate cleaning effectiveness.

Rule discovery. The prior methods are classified as follows. (1) Levelwise search, *e.g.,* [86] and [127] for mining FDs; [54] and [68] for CFDs; and [159] for MDs. (2) Depth-first search, *e.g.,* [5, 180] for FDs, [54] for CFDs, and [36, 133] for DCs using evidence sets. (3) Hybrid, *e.g.,* [152] combines levelwise search with depth-first search to mine MDs, and [140] mines CFDs by integrating FD mining and itemset mining. (4) ML-based methods, *e.g.,* inductive learning [63], reinforcement learning [56], and generative models [51].

In contrast, we treat OMR discovery as a meta-learning task for improving ML classification, leveraging Bayesian optimization to efficiently tune predicate hyperparameters without exhaustive search or traditional support-confidence measures.

Explainable AI. Existing work can be categorized as classic [121, 139, 143] and deep-learning-based [98, 158]. Classic XAI methods tend to employ simpler and interpretable models to explain ML predictions, such as local-approximation-based LIME [143], causal-reasoning-based CAPE [121], and decision trees [139]. Deep-learning-based methods, on the contrary, try to directly explain deep models via the contribution of input features, *e.g.,* DeepLIFT [158], decision boundaries, *e.g.,* Ex-TuNe [98], and neural saliency [94]. Different from these methods, OMRs employ logic and function predicates to explain ML predictions, justify the correctness of repairing values, and explain the causes of outliers. Moreover, the logic rules of OMRs can identify ugly outliers from a combination of data points, even when each individual outlier may not appear ugly on its own.

8 CONCLUSION

This paper aims to reduce the negative impact of outliers on classifiers $\mathcal{M}$ by: (1) classifying outliers as good, bad, and ugly, with only ugly ones affecting $\mathcal{M}$'s accuracy; (2) proposing OMRs to detect and fix ugly outliers using logic reasoning, ML predicates, and function predicates; (3) developing OHunt to prepare cleaned data by fixing ugly outliers; (4) introducing a meta-learning-based rule discovery algorithm; (5) providing a "deep cleaning" algorithm for recursively detecting and fixing ugly outliers with accuracy guarantees. Experiments show that OHunt is effective in practice.

Future work includes extending OHunt to regression ML models and studying outliers' impact on ML fairness and robustness.

REFERENCES

[1] 2025. Code, datasets and full version. https://anonymous.4open.science/r/code-3B65/.

[2] Mohamed Abdelaal, Christian Hammacher, and Harald Schöning. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *EDBT*. OpenProceedings.org, 499–511.

[3] Mohamed Abdelaal, Rashmi Koparde, and Harald Schöning. 2023. AutoCure: Automated Tabular Data Curation Technique for ML Pipelines. In *aiDM@SIGMOD*. ACM, 1:1–1:11.

[4] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N van Rijn, and Joaquin Vanschoren. 2018. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning* 107 (2018), 79–108.

[5] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.

[6] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[7] Charu C Aggarwal and Charu C Aggarwal. 2013. Applications of Outlier Analysis. *Outlier analysis* (2013), 373–400.

[8] Charu C Aggarwal and Saket Sathe. 2015. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter* 17, 1 (2015), 24–47.

[9] Herman Aguinis, Ryan K Gottfredson, and Harry Joo. 2013. Best-practice recommendations for defining, identifying, and handling outliers. *Organizational research methods* 16, 2 (2013), 270–301.

[10] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *SIGKDD*. 2623–2631.

[11] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. 2018. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research* 18, 234 (2018), 1–78.

[12] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[13] Sébastien M R Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. 2020. Learn2learn: A Library for Meta-Learning Research. (2020). arXiv:2008.12284 [cs.LG] http://arxiv.org/abs/2008.12284

[14] American Diabetes Association. 2021. Standards of medical care in diabetes—2021 abridged for primary care providers. *Clinical Diabetes* 39, 1 (2021), 14–43.

[15] Konrad Banachewicz and Luca Massaron. 2022. *The Kaggle Book: Data analysis and machine learning for competitive data science*. Packt Publishing Ltd.

[16] Xianchun Bao, Zian Bao, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, et al. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD (industrial track)*.

[17] V Barnett. 1994. Outliers in statistical data. *John Wiley & Sons Google Scholar* 2 (1994), 705–708.

[18] Mridula Batra and Rashmi Agrawal. 2018. Comparative analysis of decision tree algorithms. In *Nature Inspired Computing*. Springer, 31–36.

[19] Thomas Bayes. 1991. An essay towards solving a problem in the doctrine of chances. 1763. *MD computing: Computers in medical practice* 8, 3 (1991), 157–171.

[20] Sara Beery, Yang Liu, Dan Morris, Jim Piavis, Ashish Kapoor, Neel Joshi, Markus Meister, and Pietro Perona. 2020. Synthetic examples improve generalization for rare classes. In *The IEEE/CVF winter conference on applications of computer vision*. 863–873.

[21] Liron Bergman and Yedid Hoshen. 2020. Classification-based anomaly detection for general data. *arXiv preprint arXiv:2005.02359* (2020).

[22] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[23] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Annual Workshop on Computational Learning Theory*. 144–152.

[24] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–37.

[25] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* 50 (2003), 251–277.

[26] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[27] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. In *SIGMOD*. 93–104.

[28] John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*. Springer, 227–236.

[29] Adam D Bull. 2011. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research* 12, 10 (2011).

[30] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data mining and knowledge discovery* 30 (2016), 891–927.

[31] Lei Cao, Yizhou Yan, Yu Wang, Samuel Madden, and Elke A Rundensteiner. 2023. AutoDo: Automatic outlier detection. *Proc. ACM Manag. Data* 1, 1 (2023), 1–27.

[32] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In *SIGMOD*.

[33] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in Machine Learning Software: Why? How? What to do? *CoRR* abs/2105.12195 (2021). arXiv:2105.12195 https://arxiv.org/abs/2105.12195

[34] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *SIGKDD*. 785–794.

[35] Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 1. IEEE, 539–546.

[36] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[37] William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier, 115–123.

[38] Corinna Cortes. 1995. Support-Vector Networks. *Machine Learning* (1995).

[39] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[40] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[41] Yuhao Deng, Deng Qiyan, Chengliang Chai, Lei Cao, Nan Tang, Ju Fan, Jiayi Wang, Ye Yuan, and Guoren Wang. 2024. IDE: A System for Iterative Mislabel Detection. (2024), 500–503. https://doi.org/10.1145/3626246.3654737

[42] Yuhao Deng, Yu Wang, Lei Cao, Lianpeng Qiao, Yuping Wang, Jingzhe Xu, Yizhou Yan, and Samuel Madden. 2024. Outlier Summarization via Human Interpretable Rules. *PVLDB* 17, 7 (2024), 1591–1604.

[43] Rémi Domingues, Maurizio Filippone, Pietro Michiardi, and Jihane Zouaoui. 2018. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern recognition* 74 (2018), 406–421.

[44] D. Dua and C. Graff. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml Irvine, CA: University of California, School of Information and Computer Science.

[45] Richard O Duda, Peter E Hart, et al. 1973. *Pattern classification and scene analysis*. Vol. 3. Wiley New York.

[46] Simao Eduardo, Alfredo Nazábal, Christopher KI Williams, and Charles Sutton. 2020. Robust variational autoencoders for outlier detection and repair of mixed-type data. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4056–4066.

[47] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. *PVLDB* 8, 1 (2014), 61–72.

[48] Nidula Elgiriyewithana. 2024. Apple Quality Dataset. https://doi.org/10.34740/kaggle/dsv/7384155

[49] Andrew Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. 2015. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158* (2015).

[50] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, Vol. 96. 226–231.

[51] Lihang Fan, Wenfei Fan, Ping Lu, Chao Tian, and Qiang Yin. 2024. Enriching Recommendation Models with Logic Conditions. *Proc. ACM Manag. Data* (2024).

[52] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[53] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[54] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2010. Discovering conditional functional dependencies. *TKDE* 23, 5 (2010), 683–698.

[55] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugey, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE, 634–647.

[56] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. ACM, 384–398.

[57] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[58] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *SIGMOD*. ACM, 2706–2710.

[59] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NIPS*. 2962–2970.

[60] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. 2018. Scalable meta-learning for Bayesian optimization. *stat* 1050, 6 (2018).

[61] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-

learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

[62] Ronald A Fisher. 1922. On the interpretation of χ 2 from contingency tables, and the calculation of P. *Journal of the royal statistical society* 85, 1 (1922), 87–94.

[63] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.

[64] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[65] Benoît Frénay and Michel Verleysen. 2013. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems* 25, 5 (2013), 845–869.

[66] Johannes Fürnkranz and Johann Petrak. 2001. An evaluation of landmarking variants. In *ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*. 57–68.

[67] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. 1995. *Bayesian data analysis*. Chapman and Hall/CRC.

[68] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.

[69] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. 2017. Google Vizier: A service for black-box optimization. In *SIGKDD*. 1487–1495.

[70] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.

[71] Rihan Hai, Christos Koutras, Andra Ionescu, Ziyu Li, Wenbo Sun, Jessie van Schijndel, Yan Kang, and Asterios Katsifodimos. 2023. Amalur: Data Integration Meets Machine Learning. IEEE, 3729–3739.

[72] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Record* 29, 2 (2000), 1–12.

[73] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. 2022. ADBench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems* 35 (2022), 32142–32159.

[74] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE international conference on computer vision*. 1026–1034.

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*. 770–778.

[77] Shuo He, Lei Feng, and Guowu Yang. 2023. Partial-label Learning with Mixed Closed-set and Open-set Out-of-candidate Examples. In *SIGKDD*. 722–731.

[78] Zengyou He, Xiaofei Xu, and Shengchun Deng. 2003. Discovering cluster-based local outliers. *Patter Recognition Letters* (2003), 1641–1650. Issue No.9-10.

[79] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A demonstration of KGLac: A data discovery and enrichment platform for data science. *PVLDB* 14, 12 (2021), 2675–2678.

[80] Mauricio A Hernández and Salvatore J Stolfo. 1995. The merge/purge problem for large databases. *ACM Sigmod Record* 24, 2 (1995), 127–138.

[81] Aidan Hogan and Aidan Hogan. 2020. SPARQL query language. *The Web of Data* (2020), 323–448.

[82] John H Holland. 1992. Genetic algorithms. *Scientific American* 267, 1 (1992), 66–73.

[83] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2022. TabBFN: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848* (2022).

[84] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. John Wiley & Sons.

[85] Hao Huang, Hong Qin, Shinjae Yoo, and Dantong Yu. 2014. Physics-based anomaly detection defined on manifold space. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 2 (2014), 1–39.

[86] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal* (1999).

[87] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2014. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*. PMLR, 754–762.

[88] Boris Iglewicz and David C Hoaglin. 1993. *Volume 16: How to detect and handle outliers*. Quality Press.

[89] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: A review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[90] Kenji Kawaguchi, Leslie P Kaelbling, and Tomás Lozano-Pérez. 2015. Bayesian optimization with exponential convergence. *Advances in neural information processing systems* 28 (2015).

[91] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[92] Hufsa Khan, Muhammad Tahir Rasheed, Shengli Zhang, Xizhao Wang, and Han Liu. 2024. Empirical study of outlier impact in classification context. *Expert Systems with Applications* 256 (2024), 124953.

[93] Rabie El Kharoua. 2024. Students Performance Dataset. https://doi.org/10.34740/KAGGLE/DS/5195702

[94] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. 2019. The (un) reliability of saliency methods. *Explainable AI: Interpreting, explaining and visualizing deep learning* (2019), 267–280.

[95] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

[96] Edwin M Knox and Raymond T Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *VLDB*. 392–403.

[97] Sefa Kocakalay. 2020. diabets — kaggle.com. https://www.kaggle.com/datasets/sefakocakalay/diabets. [Accessed 27-04-2025].

[98] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[99] R Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Morgan Kaufman Publishing* (1995).

[100] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. 2008. Angle-based outlier detection in high-dimensional data. In *SIGKDD*. 444–452.

[101] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. *CoRR* abs/1711.01299 (2017).

[102] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning for Statistical Modeling. *PVLDB* 9, 12 (2016), 948–959.

[103] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. *CoRR* abs/1904.11827 (2019).

[104] Meng-Chieh Lee, Shubhranshu Shekhar, Christos Faloutsos, T Noah Hutson, and Leon Iasemidis. 2021. Gen2Out: Detecting and ranking generalized anomalies. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 801–811.

[105] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. 2012. Selecting classification algorithms with active testing. In *International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*. Springer, 117–131.

[106] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A study for evaluating the impact of data cleaning on ML classification tasks. In *ICDE*. IEEE, 13–24.

[107] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. 2020. COPOD: Copula-based outlier detection. In *IEEE international conference on data mining (ICDM)*. IEEE, 1118–1123.

[108] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H Chen. 2022. ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. *TKDE* 35, 12 (2022), 12181–12193.

[109] Boyang Liu, Pang-Ning Tan, and Jiayu Zhou. 2022. Unsupervised anomaly detection by robust density estimation. In *AAAI*, Vol. 36. 4101–4108.

[110] Boyang Liu, Ding Wang, Kaixiang Lin, Pang-Ning Tan, and Jiayu Zhou. 2021. RCA: A deep collaborative autoencoder approach for anomaly detection. In *IJCAI*, Vol. 2021. 1505.

[111] Defu Liu, Wen Li, Lixin Duan, Ivor W Tsang, and Guowu Yang. 2023. Noisy label learning with provable consistency for a wider family of losses. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 11 (2023), 13536–13552.

[112] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *International Conference on Data Mining*. IEEE, 413–422.

[113] Huan Liu and Hiroshi Motoda. 2007. *Computational methods of feature selection*. CRC press.

[114] Tongliang Liu and Dacheng Tao. 2015. Classification with noisy labels by importance reweighting. *IEEE Transactions on pattern analysis and machine intelligence* 38, 3 (2015), 447–461.

[115] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2020. Picket: Self-supervised Data Diagnostics for ML Pipelines. *CoRR* abs/2006.04730 (2020). https://arxiv.org/abs/2006.04730

[116] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1, 1 (2011), 14–23.

[117] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[118] Oded Maimon and Lior Rokach. 2005. *Data mining and knowledge discovery handbook*. Vol. 2. Springer.

[119] Donato Malerba. 1994. Page Blocks Classification. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5J590.

[120] Rafael Martinez-Guerra and Juan Luis Mata-Machuca. 2014. Fault detection and diagnosis in nonlinear systems. *Understanding Complex Systems, Springer International Publishing, Cham* (2014).

[121] Zhengjie Miao, Qitian Zeng, Chenjie Li, Boris Glavic, Oliver Kennedy, and Sudeepa Roy. 2019. CAPE: explaining outliers by counterbalancing. *PVLDB* 12, 12 (2019), 1806–1809.
[122] Kenta Nakai. 1991. Yeast. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KG68.
[123] Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, and Fatima Mohamad Dakalbab. 2021. Machine learning for anomaly detection: A systematic review. *IEEE Access* 9 (2021), 78658–78700.
[124] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. 2013. Learning with noisy labels. *Advances in neural information processing systems* 26 (2013).
[125] John A Nelder and Roger Mead. 1965. A simplex method for function minimization. *The computer journal* 7, 4 (1965), 308–313.
[126] A Nichol. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).
[127] Noel Novelli and Rosine Cicchetti. 2001. Fun: An efficient algorithm for mining functional and embedded dependencies. In *International Conference on Database Theory*. Springer, 189–203.
[128] Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. 2018. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *SIGKDD*. 2041–2050.
[129] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. 2021. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)* 54, 2 (2021), 1–38.
[130] Guansong Pang, Chunhua Shen, Huidong Jin, and Anton van den Hengel. 2023. Deep weakly-supervised anomaly detection. In *SIGKDD*. 1795–1807.
[131] Guansong Pang, Chunhua Shen, and Anton Van Den Hengel. 2019. Deep anomaly detection with deviation networks. In *SIGKDD*. 353–362.
[132] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.
[133] Eduardo HM Pena, Eduardo C De Almeida, and Felix Naumann. 2019. Discovery of approximate (and exact) denial constraints. *PVLDB* 13, 3 (2019), 266–278.
[134] Tomáš Pevnỳ. 2016. Loda: Lightweight on-line detector of anomalies. *Machine Learning* 102 (2016), 275–304.
[135] Eve Porcello and Alex Banks. 2018. *Learning GraphQL: Declarative data fetching for modern web apps*. " O'Reilly Media, Inc.".
[136] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. 2019. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research* 20, 53 (2019), 1–32.
[137] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased boosting with categorical features. *Advances in neural information processing systems* 31 (2018).
[138] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. 2021. Neural transformation learning for deep anomaly detection beyond images. In *International conference on machine learning*. PMLR, 8703–8714.
[139] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1 (1986), 81–106.
[140] Joeri Rammelaere and Floris Geerts. 2019. Revisiting conditional functional dependency discovery: Splitting the "C" from the "FD". In *Machine Learning and Knowledge Discovery in Databases: European Conference (ECML PKDD)*. Springer, 552–568.
[141] Shebuti Rayana. 2016. ODDS library. *Stony Brook University, Department of Computer Sciences* (2016).
[142] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.
[143] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *SIGKDD*. 1135–1144.
[144] Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.
[145] Peter J Rousseeuw and Christophe Croux. 1993. Alternatives to the median absolute deviation. *Journal of the American Statistical association* 88, 424 (1993), 1273–1283.
[146] Peter J Rousseeuw and Bert C Van Zomeren. 1990. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical association* 85, 411 (1990), 633–639.
[147] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *International conference on machine learning*. PMLR, 4393–4402.
[148] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. 2019. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694* (2019).
[149] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *SIGMOD*.
[150] Markus M. Breunig;Hans-Peter Kriegel;Raymond T. Ng;Jörg Sander. 2000. LOF: Identifying density-based local outliers. *SIGMOD Record* (2000), 93–104. Issue No.2.
[151] Saket Sathe and Charu Aggarwal. 2016. LODES: Local density meets spectral outlier detection. In *SIAM international conference on data mining*. SIAM, 171–179.
[152] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient discovery of matching dependencies. *ACM Trans. on Database Syst. (TODS)* 45, 3 (2020), 1–33.
[153] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.
[154] Burr Settles. 2009. *Active learning literature survey*. University of Wisconsin-Madison Department of Computer Sciences.
[155] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
[156] Wei Shen, Jianyong Wang, and Jiawei Han. 2014. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2014), 443–460.
[157] Tom Shenkar and Lior Wolf. 2022. Anomaly detection for tabular data with internal contrastive learning. In *International conference on learning representations*.
[158] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *International conference on machine learning*. PMlR, 3145–3153.
[159] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*. 1421–1424.
[160] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. 2016. Bayesian optimization with robust Bayesian neural networks. *Advances in neural information processing systems* 29 (2016).
[161] Ashwin Srinivasan. 1993. Statlog (Landsat Satellite). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55887.
[162] Shan Suthaharan and Shan Suthaharan. 2016. Support vector machine. *Machine learning models and algorithms for big data classification: Thinking with examples for effective learning* (2016), 207–235.
[163] Kevin Swersky, Jasper Snoek, and Ryan P Adams. 2013. Multi-task Bayesian optimization. *Advances in neural information processing systems* 26 (2013).
[164] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast anomaly detection for streaming data. In *International Joint Conference on Artificial Intelligence*. Citeseer.
[165] Roy Taylor. 2013. Type 2 diabetes: etiology and reversibility. *Diabetes care* 36, 4 (2013), 1047–1055.
[166] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Singh. 2020. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data* 7 (2020), 1–30.
[167] John W Tukey. 1977. Exploratory data analysis. *Reading/Addison-Wesley* (1977).
[168] J Vanschoren. 2018. Meta-Learning: A Survey. *arXiv preprint arXiv:1810.03548* (2018).
[169] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
[170] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*. 1096–1103.
[171] Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *SIGIR*. 200–207.
[172] Denny Vrandečić. 2012. Wikidata: A new platform for collaborative data collection. In *International Conference on World Wide Web*. 1063–1064.
[173] Fulton Wang and Cynthia Rudin. 2015. Falling rule lists. In *Artificial intelligence and statistics*. PMLR, 1013–1022.
[174] Hu Wang, Guansong Pang, Chunhua Shen, and Congbo Ma. 2019. Unsupervised representation learning by predicting random distances. *arXiv preprint arXiv:1912.12186* (2019).
[175] Jianwei Wang, Ying Zhang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2024. Missing Data Imputation with Uncertainty-Driven Network. *Proc. ACM Manag. Data* 2, 3 (2024), 1–25.
[176] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. 2020. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588* (2020).
[177] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.
[178] Kaiwen Wu, Kyurae Kim, Roman Garnett, and Jacob Gardner. 2024. The behavior and convergence of local Bayesian optimization. *Advances in neural information processing systems* 36 (2024).
[179] Yinjun Wu, James Weimer, and Susan B. Davidson. 2021. CHEF: A Cheap and Fast Pipeline for Iteratively Cleaning Label Uncertainties. *PVLDB* 14, 11 (2021), 2410–2418.
[180] Catharine Wyss, Chris Giannella, and Edward Robertson. 2001. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *DaWaK*.

[181] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. 2023. Deep Isolation Forest for Anomaly Detection. *TKDE* (2023), 1–14.

[182] Hongzuo Xu, Yijie Wang, Songlei Jian, Qing Liao, Yongjun Wang, and Guansong Pang. 2024. Calibrated one-class classification for unsupervised time series anomaly detection. *TKDE* (2024).

[183] Hongzuo Xu, Yijie Wang, Guansong Pang, Songlei Jian, Ning Liu, and Yongjun Wang. 2023. RoSAS: Deep semi-supervised anomaly detection with contamination-resilient continuous supervision. *Information Processing & Management* 60, 5 (2023), 103459.

[184] Hongzuo Xu, Yijie Wang, Juhui Wei, Songlei Jian, Yizhou Li, and Ning Liu. 2023. Fascinating supervisory signals and where to find them: Deep anomaly detection with scale learning. In *International Conference on Machine Learning*. PMLR, 38655–38673.

[185] Xiaodan Xu, Huawen Liu, Li Li, and Minghai Yao. 2018. A comparison of outlier detection techniques for high-dimensional data. *International Journal of Computational Intelligence Systems* 11, 1 (2018), 652–662.

[186] Huimin Zhao and Sudha Ram. 2008. Entity matching across heterogeneous data sources: An approach based on constrained cascade generalization. *Data & Knowledge Engineering* 66, 3 (2008), 368–381.

[187] Liang Zhao, Qingcan Li, Pei Wang, Jiannan Wang, and Eugene Wu. 2020. ActiveDeeper: A model-based active data enrichment system. *PVLDB* 13, 12 (2020), 2885–2888.

[188] Yue Zhao and Maciej K Hryniewicki. 2018. XGBOD: Improving supervised outlier detection with unsupervised representation learning. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[189] Yingjie Zhou, Xucheng Song, Yanru Zhang, Fanxing Liu, Ce Zhu, and Lingqiao Liu. 2021. Feature encoding with autoencoders for weakly supervised anomaly detection. *IEEE Transactions on Neural Networks and Learning Systems* 33, 6 (2021), 2454–2465.

[190] Xingquan Zhu and Xindong Wu. 2004. Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review* 22 (2004), 177–210.

[191] Ryszard Zieliński. 1983. PJ Huber; Robust statistics. *Mathematica Applicanda* 11, 23 (1983).

[192] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*.

A IMPLEMENTING OMR PREDICATES

We show how to implement ML and function predicates of OMRs.

A.1 Implementation of ML Predicates

We start with ML predicates, namely, $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ for outlier detection and $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ for determining influential features.

$\underline{\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})}$. We implement $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ using the tabular outlier detection models from [73, 181], and adopt the PReNet detector [184] as $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$. These ML-based models are classified as (1) unsupervised: Deep SVDD [147], REPEN [128], RDP [174], RCA [110], GOAD [21], NeuTraL [138], ICL [157] and DIF [181]; and (2) label-based: DevNet [131], PReNet [130], CART [42, 116], XGBoost [34], RoSAS [183], and SLAD [184], based on the availability of labels of outliers. We experimentally evaluate these outlier detectors using five-fold cross-validation [99] on 18 real-world outlier detection datasets, which are listed in Table 2 and Table 6. We find that PReNet performs the best compared to other unsupervised and label-based outlier detectors.

$\underline{\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})}$. We implement $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ to identify influential features A for $\mathcal{M}$. Feature A is considered influential if it is strongly correlated with the label or is widely involved in model training. We identify such features using both statistical and model-based methods, and take the intersection of the results from both.

(1) Statistical implementation of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$. We consider both categorical and continuous features. (a) For categorical features, we use Fisher's exact test [62] to evaluate the impact of differences between the categories in the feature columns and the label column. This helps us assess the correlation and importance of the categorical features to the label column, for us to select those features that have a significant impact on the label. (b) For continuous features, we use Local Interpretable Model-agnostic Explanations (LIME) [143] to assess the linear and nonlinear correlation between the continuous feature columns and the label column. We select continuous ones that are ranked high in terms of feature importance, as they are considered to have a significant impact on label. (c) For the label column, it serves as the output (target variable) in a classification task, while the categorical feature columns provide the input for $\mathcal{M}$'s predictions on labels. Thus, the label column naturally plays an influential role in the training process of model $\mathcal{M}$, which serves as a basis for evaluating $\mathcal{M}$'s performance.

We take the union of influential categorical, continuous features and labels to serve as statistically influential features A for $\mathcal{M}$.

(2) Model-based implementation of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$. We then filter influential features based on their involvement in the training process of $\mathcal{M}$. We first train $\mathcal{M}$ on all the original features and label column in the training set. Then, we iteratively remove one feature from all features and retrain the classifier $\mathcal{M}'$ using the remaining features and label column. After that, by testing the classification accuracy of $\mathcal{M}$ and $\mathcal{M}'$ on the test set and setting a threshold α, we consider features such that $\mathrm{acc}(\mathcal{M}, \mathcal{D}) - \mathrm{acc}(\mathcal{M}', \mathcal{D}') > \alpha$ as influential features for $\mathcal{M}$. Moreover, for certain downstream classifiers such as XGBoost [34], we take ranking based on, *e.g.,* importance scores in scikit-learn, as a basis for identifying influential features.

$\underline{\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)}$. KNN is used to fix the ugly outliers in the dataset. Below we detail the process of how to train and apply KNN for the correction. The training process for KNN includes selecting non-ugly outliers, constructing the training data, setting up the KNN model, and training the model. Non-ugly outliers are those captured by OMRs in Σ_d from the OLeaner algorithm. The features (attributes) of these non-ugly samples are used as input in the training data. The labels and feature data of these samples are known. For the KNN model, the number of neighbors K and appropriate distance metric d need to be set [45]. The optimal value of K can be determined via five-fold cross-validation [99]. During this process, d is chosen as Euclidean distance, Manhattan distance or Cosine distance. In our case, we use Euclidean distance by default. To further refine the selection of K and d, we treat these as discrete hyperparameters in OLeaner's process of learning Σ_f, and optimize them together using Bayesian optimization. Since KNN is an instance-based learning method, it does not require an explicit training process. Instead, it simply stores all the training samples [39] (see Section 5.1 for more details).

Moreover, we elaborate on what association rules need to be mined and explain why we use association rules to assist in label correction. More specifically, FP-growth [72] is applied to uncover strong relationships between the target attribute $t.A$ and other features $t.\bar{B}$ in the ugly outliers detected by OMRs in Σ_d. Association rules with high confidence, support, and lift are selected with default thresholds set to 0.9, $0.1|\mathcal{D}|$, and 3, since they indicate reliable correlations between $t.A$ and $t.\bar{B}$. For each ugly label outlier $t.A$, the corresponding association rules are used to impute the $\mathsf{ugly}(t.A)$ values based on the values of $t.\bar{B}$. The key advantage of using association rules to repair ugly labels is that it captures strong and interpretable patterns between features and labels in $\mathcal{D}$. These patterns are often more reliable than purely statistical or model-based imputation methods [118] in practice.

$\underline{\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})}$. The implementation of $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ includes three parts: (1) training model for entity matching; (2) entity matching process; and (3) data extraction from KB.

(1) The training process for entity matching involves using both the attributes $t.\bar{B}$ from $\mathcal{D}$ and the corresponding features from KB, such as textual descriptions, categories, and other metadata. To facilitate matching, similarity metrics such as string matching, numeric similarity, and categorical similarity, are calculated between the ugly tuple t in $\mathcal{D}$ and entities in KB [156]. ML models, such as decision trees [139], logistic regression [84] or deep learning models like Siamese networks [35], are then trained on labeled pairs of matching and non-matching entities to predict the probability that two entities represent the same real-world entity. The resulting model is used to match tuples in $\mathcal{D}$ with entities in KB, allowing for the extraction of accurate values for correcting ugly outliers in $\mathcal{D}$.

(2) Heterogeneous entity resolution (HER) refers to matching objects from two datasets that refer to the same real-world entity but may differ in structure or content. To resolve differences between $\mathcal{D}$ and KB, we use HER techniques [55]. The HER process identifies common attributes across datasets and uses them for matching. For example, if both $\mathcal{D}$ and KB contain entity names and geographical locations, these can be used to align records. HER involves the following techniques: (a) String matching: techniques such as Levenshtein distance (edit distance) or cosine similarity between string

representations of entity names. (b) Fuzzy matching: for data that may have small errors (e.g., typos in names), fuzzy matching algorithms can identify similar entities. Moreover, (c) attribute overlap: comparing shared attributes like location, date, or other categorical features to identify potential matches [80]. HER can be enhanced using ML models that learn from labeled data to predict whether two entities refer to the same real-world object.

(3) Once entities are matched, the relevant data from KB is extracted and used to correct the ugly outliers in $\mathcal{D}$. The matched entities in KB (denoted as v) provide the values for $t.A$ that need to be fixed. These values could include: (a) textual information (*e.g.,* descriptions, names, or titles), (b) numerical data (*e.g.,* price, rating, quantity) and (c) categorical data (*e.g.,* tags, categories) [186]. KB data can be accessed through query interfaces such as the following: (a) GraphQL [135] allows retrieving structured data efficiently. For example, users can query Wikidata to retrieve an entity's attributes based on its ID or name; (b) SPARQL [81] is used for querying RDF-based knowledge graphs, like Wikidata or DBpedia; and (c) APIs from sources such as Kaggle [15] or other datasets that provide structured data about entities. After successful entity matching, the model $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ uses the features extracted from KB (such as related entity descriptions, property values, etc.) to predict or impute the correct values for ugly outliers $t.A$.

Complexity. We outline complexity analysis of the ML models.

(1) The cost of $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ depends on the outlier detectors. For example, when using the SLAD outlier detector, the time complexity of $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ is typical $O(m \times n)$, where m is the number of network parameters and n is the number of training samples.

(2) The time complexity of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is $O(n \times (t^2 + l))$ when implemented using statistical methods on $\mathcal{M}$, where n is as above, l is cost of predicting each neighboring sample, and t is the feature dimension of training data. When $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is implemented using the model-based methods on SVM classifier $\mathcal{M}$, the cost of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is $O(n^2 \times t \times (t-1))$, where n and t are as above.

(3) The time complexity of $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ is $O(n^2)$ with n as defined earlier, which consists of the costs of ML methods and statistical methods. It is dominated by $O(n \log n)$ for KNN and $O(n^2)$ for association rule mining in ML methods. The time complexity is $O(n)$ for both fixing feature values and labels in statistical methods.

(4) The time complexity of $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ is $O(n \times |\mathsf{KB}| + n \times f)$, where $|\mathsf{KB}|$ is the size of external data source KB, and f refers to the average number of features (or attributes) associated with each entity in the knowledge base KB. This includes the cost of entity matching, feature extraction and model prediction. More specifically, the time complexity of entity matching is $O(n \times |\mathsf{KB}|)$, where the matching process requires comparing each sample in $\mathcal{D}$ with the entities in KB to find the most suitable match. Once we match an entity from the knowledge base KB, the time complexity of feature extraction in that entity is $O(n \times f)$. After extracting the features from KB, it is in $O(n)$ to pass these features as input to the model $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ for prediction.

A.2 Implementation of Function Predicates

We now show how function predicates $\mathsf{outlier}(D, R, t, A, \theta)$, $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ and $\mathsf{imbalanced}(D, R, t, A, \delta)$ are implemented.

$\underline{\mathsf{outlier}(D, R, t, A, \theta).}$ We implement $\mathsf{outlier}(D, R, t, A, \theta)$ directly based on its definition with the specified threshold θ, if it is possible to determine θ with reasonable accuracy by incorporating domain knowledge. If it is challenging to specify the parameter θ accurately, we replace the parameter θ with statistical metrics and implement this predicate at a statistical level. We present both methods below.

(1) θ-based implementation for $\mathsf{outlier}(D, R, t, A, \theta)$. First, we sort the values within the domain of attribute A to facilitate comparison. Then, we calculate the absolute difference between each data point and its neighbors. For the first (minimum) value and the last (maximum) value, they are only compared with the second smallest and second largest values, respectively. For middle ones (*i.e.,* those that are neither the minimum nor the maximum), the predicate $\mathsf{outlier}(D, R, t, A, \theta)$ is true if $|t.A - t_{\text{right}}.A|$ and $|t.A - t_{\text{left}}.A|$ both exceed the given threshold θ. If either $|t.A - t_{\text{right}}.A|$ or $|t.A - t_{\text{left}}.A|$ is within the threshold θ, $\mathsf{outlier}(D, R, t, A, \theta)$ is considered false. The criterion for determining outliers for the first and last elements is the same as that used for the middle tuples.

(2) θ-free implementation for $\mathsf{outlier}(D, R, t, A, \theta)$. Since it is difficult to determine the exact value of the threshold θ, we also employ statistical methods to estimate $\mathsf{outlier}(D, R, t, A, \theta)$ without specifying θ. More specifically, (a) we use the modified Z-score method [88] and the 2MAD_e statistical method [145] to identify outliers in the column for A. These methods define the range of normal values based on statistical indicators, and $t.A$ falling outside the range is considered outliers. Since these statistical methods do not require manually setting the threshold θ, the identification of outlier is not influenced by human factors. (b) Additionally, to reduce misclassification by $\mathsf{outlier}(D, R, t, A, \theta)$, we use filter fitting [145] and dist [146] techniques to model the data distribution of values within the domain of attribute A. By using the fitted data distribution, we identify $t.A$ outside the distribution as outliers, thus effectively estimating $\mathsf{outlier}(D, R, t, A, \theta)$. (c) To minimize the probability of missing outlier candidates, we take the union of the outliers detected by the modified Z-score, 2MADe, filter fitting, and dist methods. If a tuple t under attribute A falls within this union, then the predicate $\mathsf{outlier}(D, R, t, A, \theta)$ is true; otherwise, it is false.

$\underline{\mathsf{imbalanced}(D, R, t, A, \delta).}$ Along the same lines as implementing $\mathsf{outlier}(D, R, t, A, \theta)$, we implement $\mathsf{imbalanced}(D, R, t, A, \delta)$ using δ-based and δ-free methods. The former implements this predicate based on its definition and requires specifying the threshold δ, while the latter uses statistical methods and does not require specifying δ.

(1) δ-based implementation for $\mathsf{imbalanced}(D, R, t, A, \delta)$. When the parameter δ can be effectively determined through domain knowledge, we adopt δ-based methods as $\mathsf{imbalanced}(D, R, t, A, \delta)$. More specifically, we first count the occurrences of each distinct value in the domain of A. If the interpolation between the maximum and minimum counts exceeds the threshold δ, then $\mathsf{imbalanced}(D, R, t, A, \delta)$ is true; otherwise, $\mathsf{imbalanced}(D, R, t, A, \delta)$ is false.

(2) δ-free implementation for $\mathsf{imbalanced}(D, R, t, A, \delta)$. When it is difficult to effectively determine the parameter δ, we use the δ-free method to reduce the impact of δ. More specifically, we first normalize the values in the column of attribute A. Next, based on whether A is a categorical or continuous feature, we correspond-

ingly estimate the value of $\mathsf{imbalanced}(D, R, t, A, \delta)$, as follows. (a) If A is a categorical feature, we first count the number N of distinct values in the domain of A. We then compute the average number of supporting tuples for each distinct value in the domain of A, denoted as N_{mean}, by dividing the total number of tuples in the training set by N. If the number of supporting tuples for $t.A$ is less than N_{mean}, $\mathsf{imbalanced}(D, R, t, A, \delta)$ is true; otherwise, it is false. (b) If A is a continuous feature, we first bin the values in the A column at fixed intervals (*e.g.,* 0.05) and count the number of tuples supporting each bin. We then calculate the average value N_{mean} of these counts. Next, we determine which bin $t.A$ falls into and the number N of supporting tuples for that bin. If N is less than N_{mean}, $\mathsf{imbalanced}(D, R, t, A, \delta)$ is true; otherwise, it is false.

$\underline{\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)}$. We train the classifier $\mathcal{M}$ on the dataset $\mathcal{D}$ and specify a threshold λ for the loss predicate $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$. The form of loss is determined by the classifier $\mathcal{M}$. For instance, when $\mathcal{M}$ is an SVM classifier, $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ becomes the hinge loss function. More specifically, the hinge loss function is $\mathsf{hinge}(\mathcal{M}, D, t, j) = \max(0, 1 - d(t, y_t) + d(t, j))$, where y_t is the true label of sample t and $d(t, j)$ is the SVM's decision function value for sample t on label j. After training $\mathcal{M}$, if the possible number of labels for a sample is n, we compute the hinge loss functions $\mathsf{hinge}(\mathcal{M}, D, t, 1), \cdots, \mathsf{hinge}(\mathcal{M}, D, t, n)$ for the sample t across labels 1 to n. The loss function for sample t is computed as $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A) = \max(\mathsf{hinge}(\mathcal{M}, D, t, 1), \cdots, \mathsf{hinge}(\mathcal{M}, D, t, n))$. Here the threshold λ should be set to 1. If $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ exceeds 1, the decision function value for the true label y_t is less than that for a non-true label j, indicating that sample t is misclassified by $\mathcal{M}$ as label j, thus negatively impacting $\mathcal{M}$. Conversely, if $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ is less than 1, the true label corresponds to the maximum decision function value, meaning that $\mathcal{M}$ correctly classifies t as y_t, and so t does not negatively affect $\mathcal{M}$. In other words, samples with $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A) > 1$ are likely ugly outliers harming $\mathcal{M}$, while samples with $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A) < 1$ are likely to be good outliers or bad outliers with no negative impact on $\mathcal{M}$'s accuracy.

Complexity. We next analyze the cost of the function predicates.

(1) $\mathsf{outlier}(D, R, t, A, \theta)$. The time complexity of θ-based methods for $\mathsf{outlier}(D, R, t, A, \theta)$ is $O(nlog(n))$, where n denotes the number of training samples, as we only need to compare $t.A$ with the maximum value A_{max} and minimum value A_{min} in the domain of attribute A, and the cost of sorting the values under attribute A is $O(nlog(n))$. If we implement $\mathsf{outlier}(D, R, t, A, \theta)$ using θ-free methods, the time complexity is $O(n^2)$, where n is as above. The Z-score and 2MAD_e methods both have a time complexity of $O(n)$. This is because Z-score involves calculating values for each tuple in the training set, and 2MAD_e requires traversing each tuple to compute the median and MAD. In contrast, the time complexity for outlier detection using the filter fitting and dist methods is $O(n^2)$.

(2) $\mathsf{imbalanced}(D, R, t, A, \delta)$. The time complexity of δ-based methods for $\mathsf{imbalanced}(D, R, t, A, \delta)$ is $O(n)$, where n is as above. This is because counting the occurrences of each distinct attribute value can be done using a hash table in $O(n)$ time. For δ-free $\mathsf{imbalanced}(D, R, t, A, \delta)$, the complexity is also $O(n)$, as the main cost of this process is determined by the normalization, bucketing, and counting steps, each of which is $O(n)$.

(3) $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$. The time complexity of $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ is $O(n^2 \times d)$ when $\mathcal{M}$ is an SVM classifier, where n is the number of tuples in the training data $\mathcal{D}$ and d is the number of features in $\mathcal{D}$.

B COMPLEXITY OF OLeaner AND OFix

(1) OLeaner. OLeaner takes at most $O(l \times T + I \times (m^3 + K))$ time, where l is the size of the initial set of configurations, T is the time to evaluate a configuration, I is the number of iterations before stopping, and K is the number of steps for optimizing the acquisition function, often smaller than m^3. This cost consists of three parts: (1) $O(l \times T)$ is the cost of evaluating the initial set of configurations; (2) $O(I \times m^3)$ is for surrogate model training and (3) $O(I \times K)$ is for acquisition function optimization during each iteration.

(2) OFix. The time complexity of OFix is $O(|U| \times (\tau \times n + |\Sigma|) + n \times m \times l)$, where τ is the number of nearest neighbors considered in the KNN algorithm, $|\Sigma|$ is the number of rules in Σ, n is the number of tuples in $\mathcal{D}$, m is the number of attributes in $\mathcal{D}$, l is the number of vertices in KB, and $|U|$ is the number of ugly outliers to be repaired in U. This complexity includes the training cost of $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$, and the cost of the chasing process. Specifically, the training complexity of $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ is $O(\tau \times n \times |U| + n \times m \times log(m))$, while the complexity for $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ is $O(n \times m \times l + n \times r)$, where r is the average number of relevant neighbors in KB for $\mathsf{ugly}(t.A)$. The complexity for the chasing process is $O(|\Sigma| \times |U| + |U| \times log(|U|))$.

C PROOF OF PROPOSITION 1

We show that OFix is Church-Rosser. The proof has two steps.

(1) *Any chasing sequence is finite.* Intuitively, each chasing step fixes at least one ugly outlier and the set U is finite; hence so is ξ. More specifically, for any terminal chasing sequence $\xi = (\mathcal{F}_0, \ldots, \mathcal{F}_n)$ of $\mathcal{D}$ by (Σ, Γ), we can verify that $n \leq |\mathcal{D}|^2 + |\Gamma||\mathcal{D}|$, as a chase step (a) assigns a constant c from Γ or deduces fixes to $t.A$, which makes at most $|\Gamma||\mathcal{D}|$ steps; or (b) sets of two attributes equal, at most enumerating all tuple pairs ($|\mathcal{D}|^2$). Note that the fixes given by $\mathcal{M}_{\mathsf{FixA}}$ and $\mathcal{M}_{\mathsf{FixE}}$ are values (or mean/median) from either normal samples in $\mathcal{D}$ or values from external KB, while the number of values from KB is bounded by the number of outliers in $\mathcal{D}$, and the size of such values do not exceed outlier values. Thus, the chase sequence introduces limited number of values to Γ and ξ is finite.

(2) *All chasing sequences terminate at the same result.* Assume by contradiction that there exist two terminal chasing sequences $\xi_1 = (\mathcal{F}_0, \mathcal{D}_0) \Rightarrow_{\varphi_1, h_1} (\mathcal{F}_1, \mathcal{D}_1) \Rightarrow \cdots \Rightarrow (\mathcal{F}_{n_1-1}, \mathcal{D}_{n_1-1}) \Rightarrow_{\varphi_{n_1}, h_{n_1}} (\mathcal{F}_{n_1}, \mathcal{D}_{n_1})$ and $\xi_2 = (\mathcal{F}'_0, \mathcal{D}'_0) \Rightarrow_{\varphi'_1, h'_1} (\mathcal{F}'_1, \mathcal{D}'_1) \Rightarrow \cdots \Rightarrow (\mathcal{F}'_{n_2-1}, \mathcal{D}'_{n_2-1}) \Rightarrow_{\varphi'_{n_2}, h'_{n_2}} (\mathcal{F}'_{n_2}, \mathcal{D}'_{n_2})$ of $\mathcal{D}$ with (Σ, Γ), with different results. Note that $\mathcal{D}_{n_1}$ (resp. $\mathcal{D}'_{n_2}$) is generated from $\mathcal{D}$ by employing fixes in $\mathcal{F}_{n_1}$ (resp. $\mathcal{F}_{n_2}$). Since ξ_1 and ξ_2 differ, there exists a variable set $S = \mathcal{F}'_{n_2} \setminus \mathcal{F}_{n_1} \neq \emptyset$. Let $\mathcal{F}_i$ and $\mathcal{F}'_i$ be the first pair of sets including different fixes for $\mathsf{ugly}(t.A)$ in ξ_1 and ξ_2, such that $\mathcal{F}'_i$ contains a fix p_0 in T, that is, $\mathcal{F}'_{i-1} = \mathcal{F}_{i-1}$, $p_0 \in \mathcal{F}'_i \setminus \mathcal{F}_i$ and $p_0 \in \mathcal{F}'_{n_2} \setminus \mathcal{F}_{n_1}$. This first pair must exist since ξ_1 and ξ_2 chase starting with the same fix sets $\mathcal{F}_0 = \mathcal{F}'_0$, which are initialized with the same ground truth set Γ. In addition, the validated values added to Γ are incremental, with no existing value in $\mathcal{F}_l$ will

Name	Domain	#Samples	#Features	%Outliers	$\#\lvert\Sigma_d\rvert$	$\#\lvert\Sigma_f\rvert$
Apple [48]	architecture	4,000	8	49.90	33	69
DryBean [15]	architecture	13,611	16	3.84	51	125
Obesity [15]	health	1,000	6	14.30	21	55
Balita [15]	classification	120,999	3	11.42	26	61
Star [15]	astronomy	100,000	17	18.96	69	163
Online [15]	network	40,034	12	25.79	54	188
Student [93]	education	2,392	14	4.47	38	88

Table 6: The other seven real-life datasets

be updated or removed, *i.e.,* $\mathcal{F}_l \subseteq \mathcal{F}_{l+1}$ for $l \in [0, n_1]$ (see Section 5.2). This ensures that after the $i-1$ chase steps prior to $\mathcal{F}'_i$ (resp. $\mathcal{F}_i$), $\mathcal{F}_{i-1} = \mathcal{F}'_{i-1}$ must hold. Thus, $\mathcal{F}_i$ and $\mathcal{F}'_i$ are the first pair of different sets. Assume that $(\mathcal{F}'_{i-1}, \mathcal{D}'_{i-1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}'_i, \mathcal{D}'_i)$ is the chase step in ξ_2 that causes the different value in p_0. We show that $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is also a valid chasing step; thus ξ_1 is not terminal, which contradicts the assumption. Here $\varphi = X \rightarrow p_0$ is an OMR in Σ, $\mathcal{F}_{n_1+1}$ expands $\mathcal{F}_{n_1}$ with the fix $h(t').A$, and updated data $\mathcal{D}_i$ is obtained from $\mathcal{D}_{i-1}$ with $h(t').A$.

To show that $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is a valid chase step, it suffices to prove the following two properties: (1) h'_i is also a valuation of φ'_i in $\mathcal{D}_{n_1}$ and (2) each predicate p in the precondition X of φ'_i is validated by $\mathcal{F}_{n_1}$. If these properties hold, since $h(t').A$ does not appear in $\mathcal{F}_{n_1}$ and h is a valuation of φ in $\mathcal{F}_{n_1}$, then $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is a valid chase step.

Below we outline the key ideas behind the two properties. Formally, these can be verified by induction on the length of ξ_1.

(1) We show that h'_i is a valuation of φ'_i in $\mathcal{D}_{n_1}$. This is because (a) h'_i is a valuation in $\mathcal{D}_{i-1} = \mathcal{D}'_{i-1}$, since the two chasing sequences ξ_2 and ξ_1 still have the same fix values in their prior $i-1$ chase steps. (b) Dataset $\mathcal{D}_{n_1}$ is derived from $\mathcal{D}_{i-1}$ by applying fixes; more specifically, it is by fixing ugly$(t.A)$ guided by $\mathcal{F}_l$ for $l \leq n_1$, without deleting any tuples from $\mathcal{D}_{i-1}$. Hence all the tuples involved in h'_i remain existent in dataset $\mathcal{D}_{n_1}$ (despite possibly changed values). That is, h'_i is still a valuation of OMR φ'_i in dataset D_{n_1}.

(2) Each predicate p in the precondition X of φ'_i is validated in $\mathcal{F}_{n_1}$, since p is validated by $\mathcal{F}'_{i-1}$ in our assumption and $\mathcal{F}'_{i-1} = \mathcal{F}_{i-1} \subseteq \mathcal{F}_{n_1}$. More specifically, we adopt logic reasoning, and "robust" ML models and functions as predicates in precondition X to predict the correct values for fixing ugly outliers and updating $\mathcal{F}_{j-1}$. Indeed, observe the following. First, logic predicates in X remain true in D_{n_1} since their involved values are already in $\mathcal{F}_{j-1}$ and $\mathcal{F}_{j-1} \subseteq F_{n_1}$; this is because $\mathcal{F}$ is incremental, without deleting or updating any values that are already in it in the chase process; hence $\mathcal{F}_{j-1} \subseteq F_{n_1}$. Second, the parameters and values involved in the ML or function predicates of φ'_i are already added to $\mathcal{F}_{i-1} = \mathcal{F}'_{i-1}$ since $(\mathcal{F}'_{i-1}, \mathcal{D}'_{i-1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}'_i, \mathcal{D}'_i)$ is a valid chase step (see chase step in Section 5.2). In addition, since these models are trained before the chase begins, they remain unchanged throughout the chase process. Hence, if an ML/function predicate is true at chase step $i-1$ in ξ_2 and ξ_1, it remains true in step n_1 for ξ_1. That is, all predicates in X are validated in $\mathcal{F}_{n_1}$, and h'_i is a valuation of φ'_i and can be applied to $\mathcal{D}_{n_1}$.

Putting these together, we can see that the chasing sequence ξ_1 can be extended with $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$, which contradicts the assumption. Thus the chase is Church-Rosser. □

D EXPERIMENTAL DETAILS

Datasets. As shown in Table 6, we summarize the information for the remaining seven real-life datasets used in our experiments, in addition to the details provided in Table 2.

Baselines. We compared OHunt with 22 outlier detection baselines following [42, 73, 181, 182], which are integrated with outlier correction methods detailed in Exp-2. (1) Unsupervised ones in four categories: (a) Statistical: ECOD [108] (Empirical CDF functions), COPOD [107] (based on copulas), LODA [134] (projection-based), and RCA [110] (using covariance matrices and principal component analysis). (b) Distance-based: ABOD [100] (Angle-based Outlier Detection). (c) Ensemble-based: IForest [112] (with decision trees). (d) Deep leaning: DAGMM [192] (based on autoencoders and Gaussian mixture models), SLAD [184] (self-supervised by contrastive learning), GOAD [21] (generative model-based), DSADD [147] (combining deep neural networks with support vector machines), REPEN [128] (learning latent representations of data), ICL [157] and NeuTraL [138] (using contrastive learning techniques). (2) Semi-supervised: PReNet [130] (by reconstruction error), DevNet [131] (based on deep neural networks and graphical techniques), DSAD [148] and ROSAS [183] (both based on self-supervised learning and deep neural networks). (3) Supervised: CatBoost [137] (by gradient boosting trees), LGBoost [91] (both based on gradient boosting trees), and XGBoost [34] (based on efficient gradient boosting machines, GBM). (4) Rule-based ones include ID3 [42, 139] and CART [42, 116].