GUARDRAIL: Automated Integrity Constraint Synthesis From Noisy Data

Anonymous Author(s)

ABSTRACT

Data quality issues have been a long-standing challenge in the database community. Erroneous data can lead to incorrect query results, which in turn affect the credibility of the data-driven decisions. To circumvent this issue, a common practice is to discovery integrity constraints and enforce them on the data to ensure its quality. For instance, one can use constraints entailed by functional dependencies (FDs) to detect violations in the data. However, they are often insufficient to capture the nuances of the data.

In this paper, we present a novel form of integrity constraints as a program under a domain-specific language (DSL) that can be used to detect and rectify errors in the data. On top of DSL, we propose an efficient synthesis algorithm that leverages the statistical structural properties of the data to generate the sketch of the program that significantly reduces the search space and speedup the synthesis process. To demonstrate the usefulness of our approach, we evaluate it on 12 real-world datasets for error detection. Then, we show that the synthesized integrity constraints can be used to solidify ML-integrated SQL queries over 48 queries, leading to an average reduction of 87% in the error rates. Our open-source artifact, including the GUARDRAIL framework and the datasets, is available for the community to use [2].

1 INTRODUCTION

Understanding the relationships between data attributes is a long-standing challenge in data management. These relationships, often captured as data integrity constraints, play a crucial role in improving data quality and ensuring the reliability of the entire data processing pipeline. To illustrate, we consider the example below.

EXAMPLE 1.1. *Bob, a hospital administrator, aims to determine the average likelihood of patients suffering from dyspnea (i.e., shortness of breath) across different floors within the hospital, in order to optimize medical resource allocation. Using the hospital's database and a proprietary ML model purchased from a third-party vendor, Bob formulates an ML-integrated SQL query, as shown in the "**ML-integrated SQL Query**" in Fig. 1.*

However, the reliability of the query results depends heavily on the quality of the underlying data. Noisy rows in the database, such as erroneous X-ray results or incorrect disease codes, can significantly impact the model's predictions. As a result, the errors propagate through the query and potentially mislead Bob's decisions, highlighting the urgent need for methods to safeguard ML-integrated queries from such data quality issues.

In general, any queries could be vulnerable to data quality issues. To address this, researchers and practitioners have explored the use of data integrity constraints to identify and mitigate such errors. Data constraints, which take forms such as detectors, sanitizers, or unit tests [3, 26, 28], have long been employed in traditional software systems to ensure data reliability. Inspired by these advances, we propose a novel approach to synthesize and employ data integrity constraints from noisy data. It focuses on synthesizing data integrity constraints to validate and rectify data errors. These constraints are programmatically derived from the underlying data and capture the relationships between attributes in the dataset. During query execution, the synthesized constraints act as a runtime guardrail, verifying the correctness of input rows, preventing the propagation of errors, and even rectifying detected errors.

Recently, we have witnessed two emerging trends in this space. On one hand, instead of traditional search-based methods, many recent works have adopted statistical inference to discover data constraints (e.g., functional dependencies) [10, 20, 39, 43]. The adoption of statistical inference techniques naturally provides strong tolerance to noise and scalability to large datasets. On the other hand, data constraints could be inherently interpreted as programs. The discovery of these constraints can be viewed as a programming-by-examples (PBE) problem, where the goal is to synthesize a program that satisfies the input-output examples [14]. In addition to offering a program synthesis perspective, modelling constraints as programs also allows to formulate finer-grained and domain-specific relationships within the data, which cannot be expressed by simple constraints like functional dependencies.

In this paper, we follow these trends and propose GUARDRAIL. Conceptually, we formulate a DSL that models the data-generating process (DGP) and the programs under this DSL represent the integrity constraints. Our goal is to synthesize such a program from noisy data (a.k.a., input-output examples in the PBE paradigm). To achieve this, technically, we design an efficient sketch-based synthesis algorithm that leverages the statistical structural learning technique to generate a coarse-grained structure of the program. Then, we enumerate the program space under the sketch and search for the best program that satisfies the input-output examples. This best-fitting program is then used to validate data rows and rectify potential errors. To illustrate its application in solidifying ML-integrated SQL queries, we continue from Example 1.1.

EXAMPLE 1.2. *Continuing from Example 1.1, Bob utilizes GUARDRAIL to synthesize integrity constraints from the hospital database data ahead of time. Subsequently, whenever the ML model is queried, incoming data rows are vetted against these constraints. Should a data row violate the constraints, GUARDRAIL follows the standard data proecssing practice and first offers three error handling strategies:* `raise`, `ignore`, `coerce` *and* `rectify`. *Here,* `raise`, `ignore`, *and* `coerce` *align with the semantics in standard data analytics libraries, such as* `pandas`, *whereas* `rectify` *is a novel strategy that automatically corrects the data row to the most likely correct value. In this way, Bob can not only be informed by potential discrepancies in the data but also improve the reliability of the query results.*

In summary, our contributions are threefold:

- we pioneer the use of program synthesis for discovering data integrity constraints, ensuring that the queries to the database are fortified against data errors.

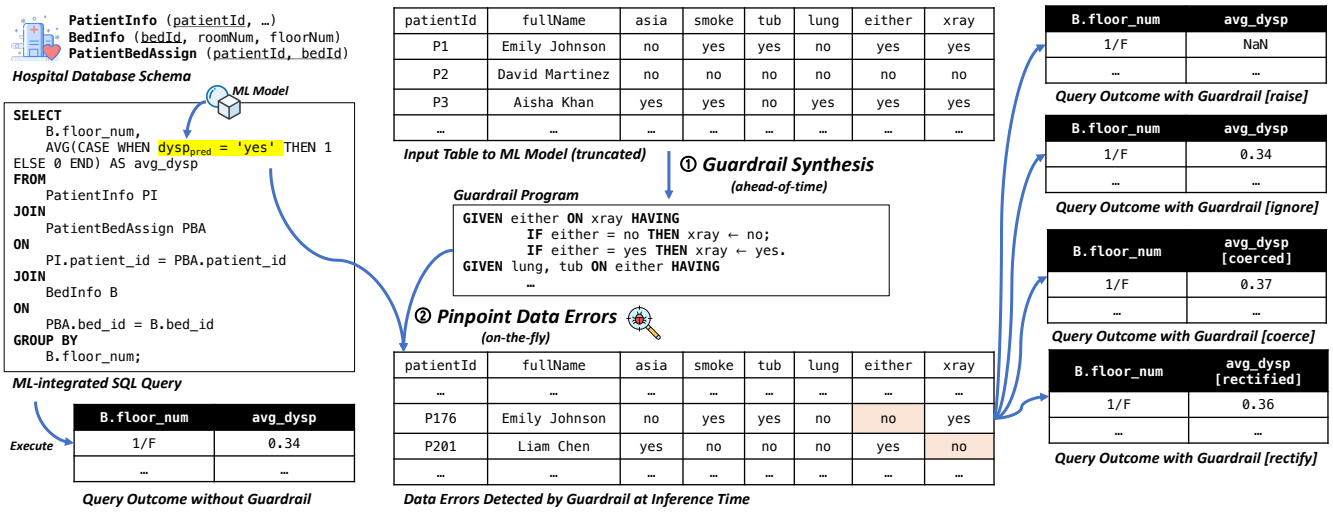

Figure 1: Example usage of GUARDRAIL in safeguarding ML-integrated SQL queries.

- we design a DSL and a sketch-based synthesis procedure that can effectively and efficiently generate data integrity constraints from noisy data.
- we evaluate GUARDRAIL on 12 real-world datasets for error detection and demonstrate its practical application in safeguarding ML-integrated SQL queries against data inaccuracies, achieving an average reduction in error rate of 87%.

Availability. Our open-source artifact, including the GUARDRAIL framework and the datasets, is available for the community to use [2].

2 FORMULATING A DSL FOR DGP

In this section, we introduce the class of DGP that we consider in this work and present a DSL to describe them.

2.1 Data-generating Process (DGP)

In statistics, it is common to model the data as a random vector X that is generated from an unseen DGP. And, the DGP can be viewed as a (probabilistic) program [29]. Visible attributes in the data are called endogenous attributes while unseen attributes are called exogenous attributes. The DPG is probabilistic when the exogenous attributes are in part of its input. Hence, each row in the dataset can be viewed as a realization of the DGP and contains I/O examples of the program. In general, DGPs can be arbitrarily complex and are often stochastic due to exogenous attributes. In this paper, we focus on discrete and deterministic DGPs for error detection.

Discrete DGP. In many data science applications, the data is often categorical. For example, consider a dataset (precisely, a database relation) D with attributes `PostalCode`, `City`, `State`, and `Country`. Here, `PostalCode` are the inputs, and `City` is the output. Given the state $\sigma = \{\texttt{PostalCode} := \texttt{94704}\}$, the program p assigns the value `Berkeley` to the variable `City` and yields the updated state $\sigma' = \{\texttt{PostalCode} := \texttt{94704}, \texttt{City} := \texttt{Berkeley}\}$. Formally, $[\![p]\!]_\sigma = \sigma'$, where $[\![\cdot]\!]_\sigma$ denotes the interpretation of the program p *w.r.t.* the state σ.

DGP For Error Detection. When the DGP is known, we can use it to detect errors in the data. Specifically, let p be the DGP program, and t be a row in the dataset (equivalently, the program state). Then, we can check whether the attributes in t is generated by p given other attributes with the assertion.

$$\forall t \in D, [\![p]\!]_t = t' \models t = t' \tag{1}$$

If the assertion does not hold, we can conclude that there is an error in the data. Continue with the above example. Let an erroneous row $t = \{\texttt{PostalCode} := \texttt{94704}, \texttt{City} := \texttt{gibbon}\}$ where "Berkeley" is corrupted as a random string "gibbon". Then, by executing the program p, we can obtain an updated state $t' = \{\texttt{PostalCode} := \texttt{94704}, \texttt{City} := \texttt{Berkeley}\}$ and it is obvious that $t \neq t'$.

2.2 Domain-Specific Language (DSL)

We present a DSL for the DGP to provides an executable, interpretable, and semantically rigorous medium to encode constraints from noisy, opaque data. The DSL, in the meantime, also allow us to extend the programming-by-example paradigm and adopt its viewpoint. Our later synthesis procedure will synthesize a program in this DSL, and the program can be used for data error detection.

Syntax. The syntax of our DSL for the DGP, as illustrated in Fig. 2, is structured into key components. $p \in$ Prog represents the whole program that consists of a series of statements. Each statement $s \in$ Stmt, represents the core program structure linking determinants to dependents based on specific conditions; the `GIVEN` clause and the `ON` clause outline determinant and dependent attributes of the generating process for the dependent attribute. Then, each statement also encapsulates a list of branches in the `HAVING` clause; $b \in$ Branch offers conditional assignments for dependent attributes; and $c \in$ Condition defines the criteria for branch validity. $a \in$ Attribute and $l \in$ Literal depict attributes and possible attribute values, respectively.

Syntax

$$p \in \text{Prog} := s^*$$
$$s \in \text{Stmt} := \texttt{GIVEN}\ a^+\ \texttt{ON}\ a\ \texttt{HAVING}\ b^+$$
$$b \in \text{Branch} := \texttt{IF}\ c\ \texttt{THEN}\ a \leftarrow l$$
$$c \in \text{Condition} := a = l \mid c\ \texttt{AND}\ c$$
$$a \in \text{Attribute} := \{a_1, \cdots, a_n\}$$
$$l \in \text{Literal} := \text{String} \cup \text{Number} \cup \text{Boolean}$$

Semantics

$$[\![l]\!] := l$$
$$[\![a]\!]_\sigma := \sigma(a)$$
$$[\![a = l]\!]_\sigma := \begin{cases} \text{true} & \text{if } [\![a]\!]_\sigma = [\![l]\!] \\ \text{false} & \text{otherwise} \end{cases}$$
$$[\![c_1\ \texttt{AND}\ c_2]\!]_\sigma := [\![c_1]\!]_\sigma \wedge [\![c_2]\!]_\sigma$$
$$[\![\texttt{IF}\ c\ \texttt{THEN}\ a \leftarrow l]\!]_\sigma := \begin{cases} \sigma[a \mapsto l] & \text{if } [\![c]\!]_\sigma = \text{true} \\ \text{skip} & \text{otherwise} \end{cases}$$
$$[\![\texttt{GIVEN}\ a^+\ \texttt{ON}\ a_j\ \texttt{HAVING}\ b^+]\!]_\sigma := [\![a_j]\!]_{[\![b^+]\!]_\sigma}$$
$$[\![s_1\ s_2]\!]_\sigma := [\![s_2]\!]_{[\![s_1]\!]_\sigma}$$

Figure 2: Formulating our DSL for the DGP and an example.

Semantics. In general, a $p \in \text{Prog}$ describes the whole DGP over a dataset while each statement $s \in \text{Stmt}$ describes the DGP for a specific dependent attribute. In our context, p is applied to an individual row in the dataset. Hence, without loss of clarity and for simplicity, given a row $t \in D$, we also use t to denote the input state for a program. Here, variables in t are the attributes in the dataset, and their values are the values of the attributes in the row. Thus, given a row $t \in D$ and a $p \in \text{Prog}$, we use $[\![p]\!]_t$ to denote the execution of p on the input state t. We present the denotational semantics of the DSL in Fig. 2. The semantics of the DSL is straightforward, and we omit the elaboration of the semantics of the DSL here.

Branch-level Loss Function. Given a dataset D and a branch b, the loss of b on D is denoted by $L(b, D)$. We adopt the 0/1 loss function as defined in [15]:

$$L(b, D) = \sum_{t \in D^b} \begin{cases} 1 & \text{if } [\![b]\!]_t \neq t \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where D^b is the set of rows in D that satisfy the condition of b. In the remainder of the paper, we use D^s and D^p to denote the union of D^b for all $b \in s$ and p, where s is a statement and p is a program, respectively. It implies that $L(b, D)$ represents the count of rows in D that do not align with the specifications of the branch. A lower loss indicates a higher quality of b.

ϵ-validity. We rule out the "bad" programs by a threshold ϵ. Specifically, given a dataset D and a $p \in \text{Prog}$, we say that p is ϵ-valid on D if

$$\forall b \in p, L(b, D) \leq |D^b|\epsilon \tag{3}$$

where $|D^b|$ is the number of rows in D that satisfy the condition of b and $\epsilon \in [0, 1)$ is a threshold. Intuitively, Eqn. 3 states that the loss of each branch in p is less than the threshold, indicating that all branches in p are applicable to the dataset D with high probability. Likewise, the statement-level thresholding is defined as follows:

$$\forall b \in s, L(b, D) \leq |D^b|\epsilon \tag{4}$$

where s is a statement. A statement satisfying Eqn. 4 is also called ϵ-valid on D.

Coverage. Given a dataset D, we first define the coverage of a branch b on D as follows:

$$\text{cov}(b, D) = \frac{|D^b|}{|D|} \tag{5}$$

where $|D^b|$ is the number of rows in D that satisfy the condition of b. Then, for a statement s, we define the coverage of s on D as the sum of the coverage of all branches in s:

$$\text{cov}(s, D) = \sum_{b \in s} \text{cov}(b, D) \tag{6}$$

Equivalently, $\text{cov}(s, D) = \frac{|D^s|}{|D|}$. Intuitively, Eqn. 5 and 6 measure the impact of a branch or statement on D, respectively. Finally, the coverage of a program p is defined as the average coverage of all statements in p.

Distinction from FD. The DSL is distinct from the traditional functional dependency (FD) in several aspects. First, FD is a declarative constraint that is defined over the entire dataset, while the DSL is a procedural constraint that is defined over individual rows. In that sense, FD itself is not capable of localizing row-level errors in the data. Second, the DSL is more expressive than FD. FD focus on the functional relationship for all possible assignments of the attributes, while the DSL allows a more flexible modeling of the DGP with some conditional branches being unconstrained.

3 RESEARCH OVERVIEW

Holistically, we aim to synthesize a program that is likely to be the ground truth DGP. In the standard programming from example (PBE) setting, given a dataset D, the synthesis problem is to synthesize a $p^* \in \text{Prog}$ that adhere the specifications entailed by the dataset D. Formally, the synthesis problem is defined as follows:

$$p^* \in \{p \mid p \in \text{Prog} \wedge \bigwedge_{b \in p} L(b, D^b) \leq |D|\epsilon\} \tag{7}$$

where $\bigwedge_{b \in p} L(b, D^b) \leq |D|\epsilon$ implies that the program is ϵ-valid on D. Similar to many program synthesis problems, there might be many programs that satisfy Eqn. 7. For instance, an empty program trivially satisfies the synthesis problem. In the following, we will dive into the details of "optimal" program of interests and present a solution to the synthesis problem.

In the following subsections, we present the key challenge of synthesizing well-formed DGP in Sec. 3.1. Then, we discuss the limitations of existing solutions on PBR-based synthesis and efforts from the database community. Finally, we present the overview of our solution in Sec. 3.2 by decomposing the synthesis problem into two sub-problems.

3.1 Challenge: Expressiveness vs. Complexity

As shown above, the primary hurdle of DGP synthesis is to balance the expressiveness and complexity of the DSL. To illustrate this hurdle, consider the following example in which we present three programs under the DSL, such that they satisfy the criterion in Eqn. 7 equally well.

Example 3.1. *Recall a dataset D with attributes* PostalCode, City, State, *and* Country. *Suppose PostalCode decides City, City decides State, and State decides Country in the ground-truth DGP. That is, the ground-truth program p^* should be expressed as follows.*

$$p^* = \mathsf{Stmt}_1\ \mathsf{Stmt}_2\ \mathsf{Stmt}_3$$

where

Stmt_1 = GIVEN PostalCode ON City HAVING $\cdots$

Stmt_2 = GIVEN City ON State HAVING $\cdots$

Stmt_3 = GIVEN State ON Country HAVING $\cdots$

where Stmt_i denotes the i-th statement in the program p. Then, a program $p_1 = \emptyset$ would satisfy Eqn. 7, where $\emptyset$ denotes an empty program. In other words, the program p_1 simply does nothing and returns the original state. It is clear that the loss of the program is zero for arbitrary datasets. However, it does not help us to detect any errors in the data. On the other hand, a saturated program p_2 that piggybacks additional unnecessary statements on p^* would also satisfy Eqn. 7.

$$p_2 = p^* \mathsf{Stmt}_4 \cdots \mathsf{Stmt}_k$$

where

$\cdots$

Stmt_4 = GIVEN PostalCode ON State HAVING $\cdots$

Stmt_5 = GIVEN PostalCode ON Country HAVING $\cdots$

$\cdots$

Stmt_k = GIVEN PostalCode, City, State ON Country HAVING $\cdots$

Conceptually, $\mathsf{Stmt}_1, \mathsf{Stmt}_2, \mathsf{Stmt}_3$ are sufficient to depict the required DGP well. However, it is non-trivial to rule out Stmt_4, $\mathsf{Stmt}_5, \cdots, \mathsf{Stmt}_k$ from p_2 in the synthesis process. For instance, Stmt_k cannot be ruled out in the absence of Stmt_3. The complex dependencies among attributes in the dataset make it challenging to identify a program that is both expressive and succinct.

$$p[\cdot] \in \mathsf{ProgSketch} := s^*$$
$$s[\cdot] \in \mathsf{StmtSketch} := \mathtt{GIVEN}\ a^+\ \mathtt{ON}\ a\ \mathtt{HAVING}\ \square$$
$$a \in \mathsf{Attribute} := \{a_1, \cdots, a_n\}$$

Figure 3: Syntax of the sketch language $\mathcal{S}$

Distinction from FD Discovery. In FD discovery, the above consideration can be seen as an analogy to the discovery of FD transitive closure or the minimal FD set. However, existing inference algorithms (e.g., Armstrong's axioms) or pruning strategies are not directly applicable to our problem given the flexibility of the DSL, which is essentially a generalized version of FDs. One notable exception is the work by Zhang et al. [43] that leverages statistical inference to discover a parsimonious set of FDs. However, as will be discussed in Sec. 6, their work has several limitations that prevent it from identifying the correct structures.

Why PBE with inductive synthesis does not fit? While we also aim to synthesize a program from examples, our setup is more challenging than standard PBE with inductive synthesis. There are three primary reasons. First, the examples are *opaque* and the input and output to the synthesizer is unknown. In other words, it is unclear which part of the data corresponds to the input or output of the DGP, which would significantly complicate the search space and beyond the capability of the existing PBE solutions. Second, the examples are *noisy* in our setting. That is, the synthesis problem has to be performed under OptSMT framework rather than the standard SMT solving, which leads to another layer of complexity. Third, the dataset we are dealing could contain tens of thousands of rows while the standard PBE solutions are often limited to a few examples. The large dataset size yields a large amount of constraints in the underlying synthesis problem, which is beyond the capability of the existing solvers. To validate our claim, we have implemented a synthesizer using OptSMT and find it does not scale to our problem setting. We report our findings in Sec. 8.3.

3.2 Synthesis from Sketch

To overcome the hurdle noted in Sec. 3.1, our key insight is that the verbosity is mainly on the GIVEN and ON clauses of the statement while the content in the HAVING clause is independent of the verbosity. Hence, if the GIVEN and ON clauses are fixed, the synthesis would be much more tractable.

Motivated by this observation and also inspired by the sketching idea in many program synthesis solutions [27, 30], we start with a high-level sketch of the program and then fill in the holes in the sketch (i.e., the HAVING clauses). In this way, we are able to offload the major complexity to the sketch learning procedure. In summary, the synthesis comprises two steps: (1) sketch learning and (2) synthesis from sketch. In the remainder of this section, we assume the sketch is given a priori and focus on the synthesis from the sketch. We will delve into the details of learning the sketch in Sec. 4.

Sketch Language and Notations. Now, we present the syntax of the sketch language, as shown in Fig. 3. In accordance with many previous works [27], on the sketch level, we only focus on discovering inter-variable dependencies among attributes in the dataset and abstracting away their concrete interactions. In particular, a program sketch remains all statements in the concrete program. However, each statement in the sketch only focuses on the GIVEN and ON clauses while leaving the branches in the HAVING clause as a hole; we express those holes as $\square$ in the sketch syntax. In the context of sketching, we extend the definition of ϵ-validity and coverage (defined in Sec. 2.2) to a program sketch: given a dataset D, a statement sketch $s[\cdot]$ is said to be ϵ-valid if and only if there exists a concretized statement $s[b^+]$ such that $s[b^+]$ is ϵ-valid. Similarly, given a dataset D, a program sketch $p[\cdot]$ is said to be ϵ-valid if and only if there exists a concretized program $p[h]$ (h is the parameter for filling the hole) such that $p[h]$ is ϵ-valid (see detailed definition in Sec. 4.1). In addition, given a dataset D, the coverage of a statement sketch $s[\cdot]$ is defined as the coverage of the its best-fit

Algorithm 1: Fill program sketch $p[\cdot]$

Input: Dataset D, Program Sketch $p[\cdot]$
Output: Concrete Program p

1 $p \leftarrow \emptyset$;
2 **foreach** $s[\cdot] \in p[\cdot]$ **do**
3 $\quad s \leftarrow$ `FillStmtSketch`$(s[\cdot], D)$;
4 $\quad$ **if** $s \neq \perp$ **then** $p \leftarrow p \cup \{s^*\}$;
5 **end**
6 **return** p;
7 **Function** `FillStmtSketch`$(s[\cdot], D)$**:**
8 $\quad b^+ \leftarrow \emptyset$;
9 $\quad$ det $\leftarrow s[\cdot]$.`GIVEN`;
10 $\quad$ dep $\leftarrow s[\cdot]$.`ON`;
11 $\quad$ Let C be the set of all warranted conditions for det according to comb(det);
12 $\quad$ **foreach** $c \in C$ **do**
13 $\quad\quad b^*[\cdot] \leftarrow$ (`IF` c `THEN` dep $\leftarrow \square$);
14 $\quad\quad l^* \leftarrow \arg\min_{l \in \text{dep}} L(b^*[l], D)$;
15 $\quad\quad$ **if** $L(s[b^*], D) < |D^{b^*}|\epsilon$ **then**
16 $\quad\quad\quad b^+ \leftarrow b^+ \cup \{b^*[l^*]\}$;
17 $\quad\quad$ **end**
18 $\quad$ **end**
19 $\quad$ **if** $b^+ \neq \emptyset$ **then return** $s[b^+]$;
20 $\quad$ **else return** $\perp$;

concretized statement $s[b^+]$. Formally, the best-fit statement $s[b^+]$ is defined as follows.

$$s[b^+] = \arg\max_{s[b^+] \in s[\cdot]} \text{cov}(s[b^+], D) \text{ s.t. } L(s[b^+], D) < |D^b|\epsilon \quad (8)$$

Synthesis Procedure. For synthesis from sketches, given a dataset D and a sketch $p[\cdot]$, we aim to synthesize a concrete program p that satisfies the criterion in Eqn. 7. We outline the procedure in Alg. 1. Given the sketch, the synthesis procedure is straightforward. For each statement $s[\cdot]$ in the sketch, Alg. 1 first initializes an empty branch list b^+ (line 8). Then, Alg. 1 extracts the determinant and dependent attributes from the `GIVEN` and `ON` clauses, respectively (lines 9–10). Then, Alg. 1 enumerates the Cartesian product for the values of the determinant attributes to constitute all warranted conditions for the determinant attributes (line 11). Next, Alg. 1 enumerates each condition c in C and initializes a branch sketch $b^*[\cdot]$ (line 13). Then, Alg. 1 picks the best-fit assignment for the dependent attribute l^* (line 14). By filling the hole in the branch sketch $b^*[\cdot]$ with l^*, Alg. 1 then checks whether the branch is ϵ-valid on D (line 15). If the branch is ϵ-valid, Alg. 1 adds the branch to the branch list b^+ (line 16). If the resulting branch list b^+ is not empty, it is filled to the hole in the statement sketch $s[\cdot]$ which is then added to the concrete program p (lines 19). Finally, Alg. 1 returns the concrete program p when all statements in the sketch are concretized (line 6). In short, the procedure between lines 7–20 in Alg. 1 aims to find the concretized statement $s[b^+]$ that is ϵ-valid on D with the highest coverage (compared to other possible ϵ-valid statements under the same sketch).

4 BUILDING DGP SKETCH

This section first formulate the sketches of interest following the discussion in Sec. 3.1. Then, we present the pipeline to synthesize the sketches from the data step by step. [1]

4.1 Characterizing Sketches of Interest

We first characterize the class of sketches that are of interest. To this end, we define two criteria to characterize the sketch for programs, namely local and global non-triviality.

In Sec. 3.2, we extend our definition on validity to sketches. However, even though a sketch is ϵ-valid, it does not necessarily mean that it is *meaningful* with respect to the DGP. For example, consider two attributes a_i and a_j in the dataset D. Even if a_i and a_j are purely independent ($a_i \perp\!\!\!\perp a_j$), we can still construct a ϵ-valid statement sketch $s[\cdot]$ with ϵ being the accuracy of random guess. However, such a trivial statement is neither informative nor helpful for detecting errors.

Definition 4.1 (Local Non-triviality (LNT)). *Given a dataset D with joint probability distribution P_D and a statement sketch $s[\cdot]$, we say that $s[\cdot]$ is locally non-trivial (LNT) if and only if $a_j \not\perp\!\!\!\perp \boldsymbol{a_k}$, where a_j is the dependent attribute in the `ON` clause, $\boldsymbol{a_k}$ is the determinant set of a_j in the `GIVEN` clause, and $\not\perp\!\!\!\perp$ denotes statistical dependence. A program sketch $p[\cdot]$ is said to be LNT if and only if all statement sketches in $p[\cdot]$ are LNT.*

Def. 4.1 regulates the notion of validity of statement sketches. Recall that the sketch encodes the inter-variable dependencies of a program. In particular, it enforces that the statement sketches must entail a statistical correlation between the dependent attribute and the determinant set. LNT implies that there exists a concretized statement $s[b^+]$ that is at least better than a random guess. For example, in the dataset described in Example 3.1, the statement sketch $s[\cdot]$ = `GIVEN PostalCode ON City` is LNT because `PostalCode` and `City` are statistically correlated.

Definition 4.2 (Global Non-triviality (GNT)). *Given a dataset D with its underlying joint probability distribution P_D and a program sketch $p[\cdot]$, we say that a LNT $p[\cdot]$ is also globally non-trivial (GNT) if each statement sketch $s[\cdot]$ in $p[\cdot]$ fulfills the following condition: for every other statement sketch $s'[\cdot]$ in $p[\cdot]$, there exists a concretized statement $s'[b^+]$ such that $\exists b \in s'[b^+]$ $s[\cdot]$ is LNT on D^b.*

GNT extends the concept of LNT to the entire program sketch and ensures that each statement sketch is deliberated in a holistic manner. The condition for GNT is that each statement sketch can be interpreted as non-trivial on the subset of the data that is conditioned on the other statement sketches. In a statistical sense, it implies that the correlation entailed by the statement sketches will not be vanished by conditioning on other statement sketches. For instance, in Example 3.1, the correlation between `PostalCode` and `State` will be vanished if we already know `City`. In other words, it requires that each statement sketch shall provide additional information that is not captured by other statement sketches. This criterion avoids constructing a program sketch that includes statement sketches, which, although individually non-trivial, do not

[1] The proofs are provided in the supplementary material [1].

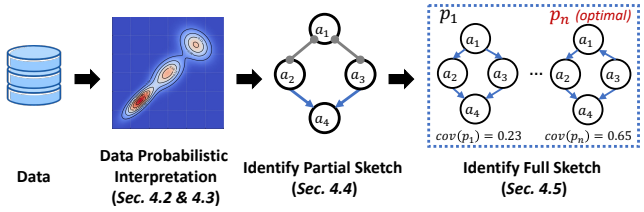

Figure 4: Sketch synthesis workflow.

jointly capture the true structure of the data when considering their interactions. To illustrate, we present the example.

Example 4.1. *Consider the dataset described in Example 3.1, and the three statement sketches:* $s_1[\cdot]$ = `GIVEN PostalCode ON City`, $s_2[\cdot]$ = `GIVEN PostalCode ON State` *and* $s_3[\cdot]$ = `GIVEN City ON State`. *Individually, each statement sketch is LNT. However, when we consider these sketches together, we would find that* `PostalCode` *is actually irrelevant to* `State` *when* `City` *is specified (i.e., on the subset of the dataset* D^b *with* b *being a condition on* `City`*). Thus, the combined program sketch* $p[\cdot] = s_1[\cdot]s_2[\cdot]s_3[\cdot]$ *is not GNT.*

Workflow. In the remainder of the section, we show how a GNT program sketch can be derived from the data, by following the workflow illustrated in Fig. 4. In Sec. 4.2 and 4.3, we provide a probabilistic interpretation of the input data through the lens of the DGP and the probabilistic graphical models (PGMs). Crucially, we show that a PGM inherently contain a GNT program sketch. In Sec. 4.4, we present several cases where the GNT program sketch can be directly extracted from the Markov equivalence classes (MECs) and in Sec. 4.5, we present an algorithm for general cases.

4.2 Data-generating Process (DGP)

In statistics, it often assumes that real-world data is generated from an (unknown) DGP. Here, we adopt the structural equation model (SEM) as the DGP to represent the data, which is defined as follows.

Definition 4.3 (Structural Equation Model [32]). *A SEM* M *consists of: (1) A directed acyclic graph (DAG)* G *with nodes* V *and edges* E*; (2) Exogenous variables* U*, representing factors outside the model; (3) Observed endogenous variables* V*, with each variable* $X \in V$ *functionally dependent on* $U_X \cup Pa_G(X)$*, where* $U_X \subseteq U$ *and* $Pa_G(X)$ *denotes the parent set of* X *in* G*; (4) Deterministic functions* $f_X \in F$*, each* $f_X : Pa_G(X) \times U_X \to X$ *computing the value of* X*.*

Each attribute in the data is represented by a random variable and the value of each attribute is derived from a deterministic generative process. The randomness of the data is captured by the unseen exogenous variables U. In SEM, the dependency among attributes can be represented by a directed acyclic graph (DAG) G (also known as Bayesian network), a popular type of probabilistic graphical models (PGMs). In the DAG, each node represents an attribute and all in-coming edges to a node represent a function generating the attribute.

Recall the definition of our DSL (defined in Sec. 2.2) and its sketch language (defined in Sec. 3.2), it is obvious that a statement, by definition, corresponds to a function in the SEM. For example, given the aforementioned dataset (Example 3.1), the statement sketch below:

$$s[\cdot] = \texttt{GIVEN PostalCode ON City HAVING } \square$$

corresponds to a function f : `PostalCode` $\to$ `City` (defined by cases) with $U_{\texttt{City}} = \emptyset$ in the SEM. Hence, one might be tempted to deduce a statement sketch from the graphical structure of the SEM (e.g., an edge from `PostalCode` to `City`). More importantly, the DAG-based representation sheds light on learning succinct sketches, as it explicitly distinguishes direct and indirect dependencies. It has potential to prioritize f : `City` $\to$ `State` over f' : `PostalCode` $\to$ `State`, because the `PostalCode` is merely an indirect dependency of `State` through `City` in the SEM.

However, despite these appealing property, the challenge stems from the theoretical difficulty of formally establishing the correspondence between the sketches and the PGMs and the practical limitations of the existing learning algorithms.

4.3 From PGMs to Program Sketches

Now, we show that the PGMs not only provide a LNT sketch, but also inherently entails a GNT (i.e., succinct) sketch.

Proposition 1. *Given a dataset* D *with its joint probability distribution* P_D *and a PGM* G*, if* P_D *is Markovian to* G*, for a variable* a_i *and its parent variables* $a_{\boldsymbol{k}}$ *in* G*, the statement sketch* $s[\cdot]$ *with the determinant set of* $a_{\boldsymbol{k}}$ *and the dependent attribute* a_i *is LNT.*

Therefore, the above proposition implies that, when the structure of the PGM is available, one can effortlessly extract a LNT program sketch according to the above proposition. In the following theorem, we further enhance the claim and show that a PGM inherently entail a GNT program sketch.

Theorem 4.1. *Given a dataset* D *with its underlying joint probability distribution* P_D *and a PGM* G*, if* P_D *is faithful to* G*, then the program sketch* $p[\cdot]$ *that is derived from* G *is GNT.*

Proof. Without loss of generality, let $s[\cdot]$ be an arbitrary statement sketch in $p[\cdot]$. In the context of the PGM, $s[\cdot]$ corresponds to one or a few directed edges from $a_{\boldsymbol{k}}$ to a_j in G, where $a_{\boldsymbol{k}}$ is the determinant attributes and a_j is the dependent attribute in $s[\cdot]$. By Proposition 1, $s[\cdot]$ is locally non-trivial. Therefore, we have that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$. To show that $s[\cdot]$ is globally non-trivial, we need to show that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$ on any subset of the data that is conditioned on other statement sketches in $p[\cdot]$ (i.e., D^b where b is a condition on other statement sketches in $p[\cdot]$). That is, the above condition is equivalent to showing that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid b$, where b is the condition. Since we are to show that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$ on any possible b, we can further rewrite the condition as $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid a_{\boldsymbol{z}}$ where $a_{\boldsymbol{z}}$ is any set of determinant attributes that are comprised by $p[\cdot]$. Since P_D is faithful to G, we have that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid a_{\boldsymbol{z}}$ for any $a_{\boldsymbol{z}}$. Therefore, we have that $s[\cdot]$ is globally non-trivial. □

Theorem 4.1 allows us to directly derive a GNT program sketch from PGM. However, the full structure of PGM is not practically learnable from the data. Next, we show that the MECs can be used to partially deduce a GNT program sketch.

4.4 Characterizing Practically Learnable Sketches from MECs

Due to the limitations of the existing structure learning algorithms, the full structure of the PGM is not practically learnable from the data. In this section, we aim to understand under what conditions we can still (partially) deduce a GNT program sketch in practice. Given the identifiability issue of the PGM [32], we cannot uniquely identify the PGM from the data distribution in general. Instead, we can learn up to the Markov equivalence class (MEC) of the PGM. Compared to the original DAG, the MEC is a summary of multiple possible DAGs given the data, which is defined as follows.

Definition 4.4 (Markov Equivalence Class [32]). *Given a joint probability distribution P, the MEC is the set of DAGs that are Markov equivalent under P, denoted as $[G]_P$ and the ground-truth DAG G is within $[G]_P$ (abbreviated as $[G]$).*

Typically, a MEC consists of multiple DAGs that are indistinguishable from each other given the data. That is, one cannot intermediately extract the sketch from the MEC as some edge directions are ambiguous in the MEC. Below, we present some cases of statement sketches with a unique mapping in MECs.

Proposition 2. *Given a LNT statement sketch $s[\cdot]$ from a MEC $[G]$, with determinant set $a_{\boldsymbol{k}}$ and dependent attribute a_j, if a_j is adjacent to every attribute in $a_{\boldsymbol{k}}$ and each attribute in $a_{\boldsymbol{k}}$ is non-adjacent to the others, then in every G of $[G]$, there is a directed edge from each attribute in $a_{\boldsymbol{k}}$ to a_j. And, no other directed edges exist to a_j in every G of $[G]$.*

Proof. We first prove the presence of directed edges from each attribute in $a_{\boldsymbol{k}}$ to a_j in every DAG G within the MEC $[G]$ by contradiction. Assume that for some pair of attributes $a_l, a_m \in a_{\boldsymbol{k}}$, the triplet a_l, a_j, a_m does not form a v-structure in $[G]$. This means that we do not have the configuration $a_l \rightarrow a_j \leftarrow a_m$. According to the definition of faithfulness, the absence of a v-structure implies that for any set of attributes $a_{\boldsymbol{z}} \subset D$ that renders a_l and a_m conditionally independent, a_j must not be in $a_{\boldsymbol{z}}$. However, by the definition of a locally non-trivial statement sketch, we have that a_l is not conditionally independent of a_m given a_j, denoted as $a_l \not\perp\!\!\!\perp a_m \mid a_j$. This is a contradiction to the assumption that a_l, a_j, a_m do not form a v-structure, as the presence of a_j in the conditioning set should not affect the conditional independence of a_l and a_m if they do not form a v-structure. Therefore, the only consistent configuration under the MEC $[G]$ is that a_l, a_j, a_m must form a v-structure, which implies that there is a directed edge from a_l to a_j and from a_m to a_j in every DAG G within the MEC $[G]$.

Then, we prove the absence of other directed edges to a_j in every DAG G within the MEC $[G]$ by contradiction. Assume that there exists a directed edge from $a_q \notin a_{\boldsymbol{k}}$ to a_j in some DAG G within the MEC $[G]$. $a_q \rightarrow a_j \leftarrow a_m$ (where $a_m \in a_{\boldsymbol{k}}$) forms a v-structure in G. This implies that $a_q \not\perp\!\!\!\perp a_m \mid a_j$. However, by the definition of statement sketch, it is clear that $a_{\boldsymbol{k}}$ are the only determinant set of a_j. Therefore, the dependency between a_q and a_m must be mediated by a_j. That is, $a_q \perp\!\!\!\perp a_m \mid a_j$. This is a contradiction to $a_q \not\perp\!\!\!\perp a_m \mid a_j$. □

Proposition 3. *Consider two conjunctive LNT statement sketches $s_1[\cdot]$, $s_2[\cdot]$. Let $a^1_{\boldsymbol{k}}$ and $a^2_{\boldsymbol{k}} = \{a_i\}$ be the determinant set and a_i and a_j the dependent attributes for $s_1[\cdot]$ and $s_2[\cdot]$, respectively. Then, in every G of $[G]$, there is a directed edge from a_i to a_j, and no other directed edges to a_j.*

Proposition 4. *Consider two conjunctive LNT statement sketches $s_2[\cdot]$, $s_3[\cdot]$. Let $a^2_{\boldsymbol{k}} = \{a_i\}$ and $a^3_{\boldsymbol{k}} = \{a_j\}$ be the determinant set and a_j and a_l the dependent attributes for $s_2[\cdot]$ and $s_3[\cdot]$, respectively. If $s_2[\cdot]$ satisfies Proposition 2, then in every G of the MEC $[G]$, there is a directed edge from a_j to a_l, and no other directed edges to a_l.*

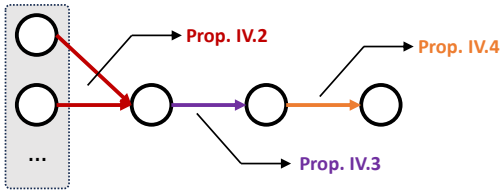

Figure 5: Practically learnable local structures *w.r.t.* Proposition 2, 3, and 4.

We prove Proposition 2 above and both Proposition 3 and 4 are direct corollary of Proposition 2. We present a visual interpretation of these propositions in Fig. 5. Intuitively, Proposition 2 states that when a statement sketch $s[\cdot]$ has more than one determinant attribute, it is unique in the MEC and can be directly extracted. Proposition 3 and 4 further claims that all its descendant statement sketches can be uniquely identified in the MEC. Otherwise, additional post-processing is needed to identify a statement sketch, as will be shown in Sec. 4.5.

4.5 Extract Program Sketches from the MEC

Algorithm 2: Synthesize optimal program p from $[G]$

Input: Dataset D, MEC $[G]$
Output: Program p

$p^* \leftarrow \emptyset$;
foreach $G \in [G]$ **do**
  $p[\cdot] \leftarrow \emptyset$;
  foreach $a_j \in D$ **do**
    $\text{pa}_j \leftarrow \text{Parent}(a_j, [G])$;
    if $pa_j \neq \emptyset$ **then**
      $s[\cdot] \leftarrow$ `GIVEN` pa_j `ON` a_j `HAVING` □;
      $p[\cdot] \leftarrow p[\cdot] \cup \{s[\cdot]\}$
  end
  $p \leftarrow p[s^*]$ according to Alg. 1;
  if $cov(p, D) > cov(p^*, D)$ **then**
    $p^* \leftarrow p$;
  end
end
return p^*;

While we have shown that the MEC can be used to directly extract the statement sketches in many cases, there are instances

where multiple directions are possible, which requires an enumerative search procedure. We outline the procedure for optimal program synthesis from the MEC $[G]$ in Alg. 2. Here, we need to find the "optimal" one out of many feasible sketches. We outline the procedure for optimal program synthesis from the MEC $[G]$ in Alg. 2. Specifically, we enumerate all DAGs within the MEC $[G]$ (line 3; subject to a maximal enumeration of DAGs, omitted in Alg. 2 for brevity) and synthesize the program p from each DAG G (lines 4–10). Then, we select the program p with the highest coverage as the optimal program p^* (lines 11-13). We consider the coverage of a program p on a dataset D as the fitness metric because it indicates the "discriminative power" of p and helps distinguish the best program from suboptimal ones.

The exhaustive search in Alg. 2 is simple but practical because the number of DAGs in the MEC is usually moderate compared to the total number of DAGs in the search space. Hence, while it is possible to further optimize it with sophisticated search strategies (e.g., pruning, hierarchical search) to reduce the computational cost, the enumeration only takes 1.7% of the total time in our experiments and we leave it as future work.

4.6 Structure-invariant Transformation

The complexity of real-world DGPs makes it challenging to accurately learn the PGM from real-world data (e.g., curse of dimensionality, data sparsity). Hence, instead of learning directly from the raw data, we anticipate to learn the PGM from an "analysis-friendly" data and employ the tactic in [43].

Definition 4.5 (Auxiliary Distribution). *Given a dataset D with joint probability distribution P_D for its attributes, we define a random vector $\mathbb{I}$. For any two rows $t_1, t_2 \sim P_D$, the k-th element, $\mathbb{I}_k$, is 0 if $t_1(a_k) \neq t_2(a_k)$ and 1 otherwise.*

As shown in Def. 4.5, $P_{\mathbb{I}}$ is constructed by recasting the equality constraint between two rows in the raw data D into the constant constraint on the binary random vector $\mathbb{I}$ to alleviate the sparsity issue. Since $\mathbb{I}_k$ is purely determined by a_k, the PGM of $P_{\mathbb{I}}$ is the same as that of P_D (see proof in appendix).

5 APPLICABILITY IN MACHINE LEARNING

The integrity constraints, in general, are capable of improving the quality of the data as well as the performance of all downstream applications (e.g., querying, machine learning). In this work, we follow the same trend of this line of research [10, 39] and focus on the applicability of the integrity constraints in ML-integrated SQL queries. In this section, we will conceptually and empirically justify why they are beneficial in this context.

First, following the setup in Fig. 1, we consider a ML model that predicts the dysp (whether a patient is short of breath) based on a list of attributes with $X, Y, \ldots$ and suppose there exists an integrity constraint such that IF $X = x$ THEN $Y \leftarrow y$. Given the deterministic relationship between $X = x$ and $Y = y$, the ML model $f : \langle X, Y, \ldots \rangle \rightarrow$ dysp could be ill-posed because the model may learn a "shortcut" by directly predicting dysp from X without considering Y when $X = x$. As a result, the model may not generalize well to data that deviates from the constraint. In contrast, by incorporating the integrity constraint into the ML model, the users are aware of the potential pitfalls of the model and can make informed decisions on how to interpret predictions.

Table 1: The effectiveness on error and mis-prediction detection across datasets.

Dataset ID	1	2	3	4	5	6	7	8	9	10	11	12
# Errors	3377	1419	35	19	6	48	124	521	444	1404	808	2591
# Mis-pred	426	336	2	5	5	14	14	321	25	33	41	383

Second, we empirically studied the mis-predictions from ML models on 12 datasets with randomly introduced errors in Table 1. As shown in Table 1, we observe that a considerable proportion of detected data errors lead to mis-predictions (0.24 on average; up to 0.83 in Contraceptive Method dataset) with a Spearman's rank correlation coefficient of 0.947 (p-value $2.91 \times 10^{-6} < 0.05$) between the number of errors and the number of mis-predictions. This indicates that the errors in the data are highly correlated with the mis-predictions of the ML models. By incorporating the integrity constraints into the ML models, we anticipate to mitigate these error-induced mis-predictions. Given these observations, we focus on its application in ML-integrated SQL queries and make it an important research question to answer in our evaluation.

6 REVISITING EXISTING SOLUTIONS

We have shown that the statistical structure of the data can be exploited to synthesize its internal integrity constraints. We rigorously proved its correctness and optimality in the theoretical sense. In this section, we revisit existing data profiling solutions that also leverage the statistical structure of the data, and provide a discussion on the distinction of our approach.

The most relevant work is FDX [43] that offers a unique perspective on discovering FDs from a statistical perspective. FDX leverages a linear autoregressive matrix learned from data to identify FDs. During this procedure, FDX makes some assumptions that may not hold in practice. First, FDX assumes that the (transformed) data (see details in Sec. 4.6) is generated from a linear additive distribution. In other words, it assumes that each attribute in $P_{\mathbb{I}}$ is generated in the following form $\mathbb{I}_k = \frac{1}{|Pa_G(a_k)|} \sum_{a_i \in Pa_G(a_k)} \mathbb{I}_i + \eta_k$ where $Pa_G(a_k)$ is the parent set of a_k. In the sense, $\mathbb{I}_k$ is subject to the count of ones in the parent set of a_k and an additive noise η_k. However, recall that the real auxiliary distribution $P_{\mathbb{I}}$ is a binary-valued distribution. It is easy to see that the formulated linear non-Gaussian distribution deviates from the real distribution. Furthermore, the assumption on the additivity of the noise term η_k is also questionable. Consider a simplified relational schema $R = \{a_1, a_2\}$ with an FD $a_1 \rightarrow a_2$ and the corresponding auxiliary distribution $P_{\mathbb{I}}$. We have $\mathbb{I}_2 = \mathbb{I}_1 + \eta_2$. Since η_k is additive in the functional form, we have $\eta_k \perp\!\!\!\perp \mathbb{I}_1$. However, this is not true in the real distribution. In fact, when $\mathbb{I}_1 = 0$, we have $P(\eta_2 = -1 \mid \mathbb{I}_1 = 0) = 0$ (as $\mathbb{I}_2 \in \{0, 1\}$) whereas $P(\eta_2 = -1 \mid \mathbb{I}_1 = 1) \neq 0$. Therefore, η_2 is dependent to $\mathbb{I}_1$ and the additivity assumption is violated. We follow the same perspective as FDX to learn the program sketch from the statistical structure of the data. However, our approach guarantees the correctness under realistic assumptions. We do not make any assumptions on the functional form of the data and directly learn the program sketch from the MEC. In Sec. 8.1, we will empirically compare our approach with FDX.

We are also aware of representative data profiling tools that follows a similar technical route. For example, CORDS [20], SCODED [39], and Conformance Constraint [10] start with statistically modeling the data and then infer the constraints accordingly. For CORDS, it extracts the pairwise correlation between attributes and then infers the FDs. As a result, it is not able to capture the redundancy and maintain a concise set of FDs. SCODED requires the user to specify the statistical constraints and provide an efficient way to detect and locate top-k violations. Our approach, on the other hand, complements SCODED by automatically inferring similar constraints from noisy data and convert to hard constraints (in contrary to the statistical constraints in SCODED) for error detection. It is possible to integrate SCODED with our approach to offer a more diverse perspective of data errors. Conformance Constraint is a recent work that learns the arithmetic constraints on numerical attributes and can be used in conjunction with our approach that focuses on the categorical attributes.

7 IMPLEMENTATION

We implement Guardrail in Python with 4.4K lines of code at [2]. Now, we describe the key implementation optimizations.

Synthesis Optimizations. To accelerate the synthesis procedure, we adopt three implementation-level optimizations. First, we take the circular shift trick [43] for efficiently sampling from the auxiliary distribution and employ the vectorization to further speed up the computation. Second, we adopt a statement-level cache mechanism to avoid redundant concretizations incurred by different DAGs in the MEC. Third, we adapted a highly-efficient Julia implementation of PDAG (a generalization of MEC) enumeration [36] to enable the enumeration operation in Alg. 2.

Error Handling Schemes In accordance to the error handling schemes in Python `pandas` package, we implement three error handling schemes in Guardrail: ❶ `raise`: raise an error when encountering an error, ❷ `ignore`: ignore the error and continue the execution, and ❸ `coerce`: replace the error with a special value, such as `NaN`. We also implement an additional scheme, ❹ `rectify`, to rectify the data errors by replacing the erroneous values with the correct ones entailed by the synthesized program. This `rectify` scheme delivers a lightweight solution to improve the accuracy of ML-integrated SQL queries, as will be shown in Sec. 8.2; in contrast, previous relevant works [10] do not provide such a capability.

Query Execution and ML Training Due to the nature of Guardrail, we need to intercept predictions made by the ML model and then enforce input error checking. However, off-the-shelf database with ML integration (e.g., SQL Server) does not support such interception. Hence, we spend considerable engineering effort to implement a SQL query executor with `pandas` ecosystem that supports the integration. Specifically, we first parse the SQL query and extract the ML model and the input attributes. Then, we feed the input rows to Guardrail and pass them to the ML backend to make predictions. Finally, we replace the ML-related expressions in the SQL query with the predictions and execute the query accordingly. We implement several standard query optimization techniques, such as predicate pushdown, to improve the performance of the query execution. As a research prototype, our query executor only supports a subset of SQL features. For example, it does not natively support the `JOIN` operation. Currently, one can use the materialized views to pre-compute the results and use our query executor over multiple tables. The design of Guardrail is agnostic to the choice of the ML model. In our implementation, we adopt a popular AutoML library, `autogluon` [8], to perform end-to-end ML model training and inference. It trains various ML models (NN, tree-based models, etc.) and create an ensemble to achieve the best performance.

Table 2: Dataset description

ID	Dataset Name	Category	# Attributes	# Rows
1	Adult	Demographic	15	48842
2	Lung Cancer	Medical	5	20000
3	Cylinder Bands	Manufacturing	40	540
4	Diabetes	Medical	9	520
5	Contraceptive Method Choice	Demographic	10	1473
6	Blood Transfusion Service Center	Medical	4	748
7	Steel Plates Faults	Manufacturing	28	1941
8	Jungle Chess	Game	7	44819
9	Telco Customer Churn	Business	21	7043
10	Bank Marketing	Business	17	45211
11	Phishing Websites	Security	31	11055
12	Hotel Reservations	Business	18	36275

8 EVALUATION

We evaluate our system on both synthetic and real-world datasets and we aim to answer the following questions:

- **RQ1: Constraint for Error Detection.** How effective is Guardrail in discovering constraints and detecting data errors?
- **RQ2: Application on ML-integrated Queries.** How effective is Guardrail in rectifying data errors and consequently improving the accuracy of ML-integrated SQL queries?
- **RQ3: Ablation Study.** How do different components and design choices of Guardrail affect the overall performance?

Below, we first describe the experimental setup and then present the results for each research question in turn.

Setup. To our best knowledge, there are no publicly available datasets that contain such ground truth data constraints and annotated data errors. Therefore, we first use 12 real-world datasets and inject data errors into them. Some datasets (e.g., Adult) are chosen due to their use in prior research on functional dependencies. Others are selected from OpenML-CC18 [4], a benchmark suite for tabular ML, containing sufficient categorical columns.[2] All experiments are conducted on a server with an AMD Threadripper 3970X CPU and 256 GB memory.

8.1 RQ1: Constraint for Error Detection

In this RQ, we aim to understand whether the constraints generated by Guardrail can effectively detect data errors.

Baselines. We primarily compare Guardrail with baselines for functional dependency discovery, including TANE [19], CTANE [9] and FDX [43]. We also tentatively studied the applicability of popular PBE systems in our setting, such as FlashFill [13], Foofah [21]. However, they are not designed for noisy data and do not support constraints expressed in our DSL. Therefore, we implement a baseline using the `OptSMT` solver [5] at our best effort, which searches

[2] See detailed data sources in [1].

Table 3: The effectiveness on error detection. "-" indicates an error (e.g., out-of-memory) during the execution.

Dataset	Metric	GUARDRAIL	TANE	CTANE	FDX
1	F1	**0.356**	0.197	0.023	0.020
	MCC	**0.389**	0.329	0.030	0.008
2	F1	**0.411**	0.286	0.165	0.049
	MCC	**0.410**	0.406	0.251	0.102
3	F1	0.333	-	**0.594**	-
	MCC	**0.355**	-	NaN	-
4	F1	0.061	NaN	**0.675**	0.612
	MCC	-0.023	NaN	**0.420**	0.291
5	F1	0.065	NaN	0.188	**0.377**
	MCC	**0.161**	NaN	-0.069	0.095
6	F1	**0.723**	0.636	NaN	0.374
	MCC	**0.684**	0.609	NaN	-0.321
7	F1	0.065	0.061	0.138	**0.220**
	MCC	**0.170**	0.069	-0.046	-0.021
8	F1	**0.065**	NaN	NaN	0.020
	MCC	**0.182**	NaN	NaN	NaN
9	F1	**0.378**	0.235	0.076	0.069
	MCC	**0.477**	0.360	0.037	0.036
10	F1	**0.051**	0.032	0.018	0.019
	MCC	0.055	**0.127**	-0.004	-0.002
11	F1	**0.139**	-	-	0.037
	MCC	**0.121**	-	-	0.019
12	F1	**0.139**	0.041	0.019	0.020
	MCC	0.130	**0.144**	NaN	0.002

Table 4: Processing time for offline synthesis.

Dataset ID	#1	#2	#3	#4	#5	#6
# Attr.	15	5	40	9	10	4
Total Time (s)	665	607	1205	690	605	604
Dataset ID	#7	#8	#9	#10	#11	#12
# Attr.	28	7	21	17	31	18
Total Time (s)	604	614	1376	820	1227	1301

for a program that minimizes the number of violated I/O examples. The `OptSMT` baseline is available in our codebase. However, it is worth noting that our program size is significantly larger than that of traditional PBE tasks and the `OptSMT` baseline is not scalable to our setting. As a result, we do not include it in the comparison and provide theoretical analysis in the ablation study.

We apply the baselines to discover constraints from an error-free split of the dataset and to detect data errors in the error-injected split. It is worth noting that some attributes are inherently unaffected by any constraints, making certain errors undetectable. Since the ground-truth constraints are unknown, we ensure a fair comparison by randomly injecting data errors into the datasets at a fixed error rate of 1% (or slightly higher for datasets with fewer rows; capped at 30 errors). We then evaluate the baselines by its F1 score and MCC (Matthews Correlation Coefficient) score, which provide a balanced view of the precision and recall of the error detection. We report the results in Table 3. It is clear that GUARDRAIL shows strong performance compared to other baselines and ranks first in 17 out of 24 comparisons (12 datasets × 2 metrics). In contrast, TANE and CTANE frequently overfit the data and detect many overly restrictive constraints, leading to lower performance. FDX is more effective than TANE and CTANE. However, due to the improper assumptions on the data distribution, FDX faces ill-conditioned matrix inversion on #3 dataset and treats all rows as errors on #8 dataset. This is because all these methods are primarily designed for knowledge discovery rather than error detection. GUARDRAIL, on the other hand, is specifically designed on a robust statistical foundation to handle noisy data. On #4 dataset, GUARDRAIL leads to a sub-optimal performance. This is because the dataset is relatively small and the statistical signal often vanishes under the small sample size, which makes our synthesis algorithm less effective in identifying the true structure from the data.

The computational overheads of constraint synthesis are proportional to both the number of DAGs in the MEC and the number of attributes. Therefore, we report these metrics in addition to the total processing time in Table 4. For datasets with fewer than 10 attributes, the processing time is 624.0 ± 34.1 seconds. Generally, datasets with more attributes take longer to process. However, some datasets, such as dataset #1, have lower processing times despite higher attribute counts. This is due to the relatively simple structure of DAGs in the MEC and the effectiveness of our statement-level cache mechanism in reducing overhead. Even for the most challenging dataset, the processing time remains manageable at 1,376 seconds. Since the synthesis process is a one-off effort for each dataset, the processing time is affordable in practice.

Answer to RQ1. GUARDRAIL offers a unique and effective solution for discovering constraints and detecting data errors that cannot provided by traditional FD discovery methods.

Table 5: The effectiveness on mis-prediction detection. "-" denotes there is no missed data errors. P is $\frac{\text{\# Detected Mis-pred.}}{\text{\#Total Detected Data Errors}}$ and R is $\frac{\text{\# Missed Mis-pred.}}{\text{\#Total Missed Data Errors}}$

ID	1	2	3	4	5	6	7	8	9	10	11	12
#Mis-pred.	426	336	2	5	5	14	14	321	25	33	41	383
P	.13	.24	.06	.26	.83	.29	.11	.62	.06	.02	.05	.15
R	-	.00	-	-	-	-	-	-	-	.00	.00	.00

8.2 RQ2: Application on ML-integrated Queries

In this RQ, we explore the application of GUARDRAIL on ML-integrated SQL queries. We first introduce the setup of experiments in the RQ. Then, we explore the relationship between data errors (detected by GUARDRAIL) and mis-predictions made by the ML model. Finally, we apply GUARDRAIL in an end-to-end setting with 48 ML-integrated SQL queries to evaluate its ability on rectifying data errors and improving the accuracy of the queries. In addition, we also provide a case study in the supplementary material [1].

Setup. In line with the configuration in [10], we split the dataset into training and test sets and apply GUARDRAIL on the training split to synthesize constraints and then apply the learned constraints on the test split to detect data errors. In this section, we focus on errors that are caused by the integrity constraints to isolate the impact of undetectable errors.

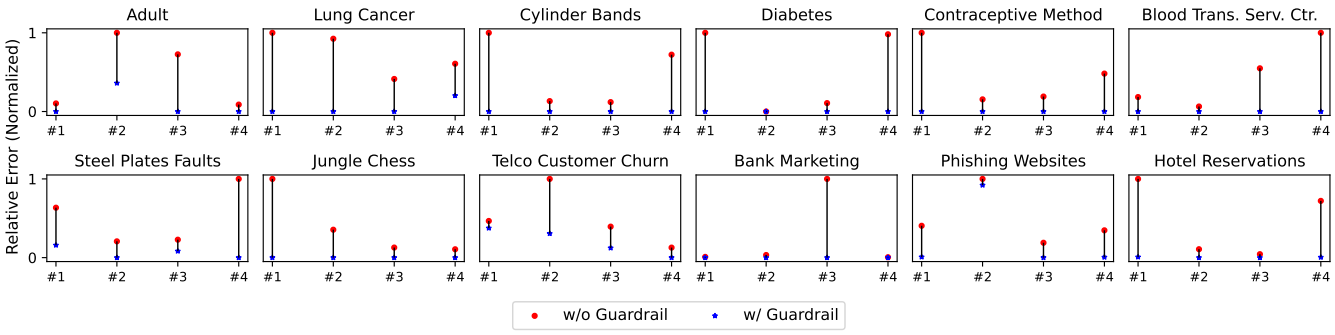

Figure 6: Effectiveness on rectifying data errors.

ML-Integrated SQL Queries. For each dataset, one author manually creates four ML-integrated SQL queries with varied complexity (four queries each; 48 queries in total). The remaining authors cross-check the queries and ensure that they are meaningful (see query details in [2]). Query results on the clean data are deemed as the ground truth. We then compare the accuracy of different query execution modes: ❶ the vanilla query execution mode (i.e., w/o GUARDRAIL), ❷ the GUARDRAIL-augmented query execution mode with data errors being rectified by our synthesized constraints. However, it is worth noting that different queries have different base units of values. For example, query #1 may aggregate the average of the label by "`CASE WHEN label=1 THEN 1 ELSE 0`" clauses, while query #2 may directly count the number of rows with label 1. Therefore, we first compute L_1 distance between the query outcomes over the clean data and the error-injected data as the *absolute error*. Next, we calculate the *relative error* by dividing the absolute error by the L_1 norm of the clean data query outcome. Finally, we normalize the relative error by min-max normalization such that all different queries have the same scale of error.

As studied in Sec. 5, data errors can lead to mis-predictions by the ML model. We further investigate how would the data errors detected by GUARDRAIL affect the predictions made by the ML model. As shown in Table 5, the data errors detected by GUARDRAIL are indeed the root cause of many mis-predictions made by the ML model. Moreover, while GUARDRAIL has a slightly lower recall, we find that, if a data error is missed by GUARDRAIL, it is not likely to lead to a mis-prediction. Surprisingly, none of the missed data errors lead to mis-predictions. This observation further promotes the real-world applicability of GUARDRAIL.

For ML-integrated queries, we report the results in Fig. 6 where red dots represent the relative error of the query over the data with data errors and blue dots represent the relative error of the query over the data with errors being rectified by GUARDRAIL. Overall, we observe that GUARDRAIL improves the accuracy for all queries. In particular, it delivers an average reduction of 0.87 ± 0.25.

As expected, GUARDRAIL introduces a runtime overhead compared to the vanilla query execution mode. We report the runtime overheads and their breakdown in Table 6. Overall, the GUARDRAIL runtime is dominated by the number of data points to be checked and also the complexity of the program (e.g., the number of statements and the number of attributes involved). The average runtime overhead is 0.332 seconds, which is modest in practice. Compared to the ML model inference time, the runtime overhead is lightweight and sometimes even negligible. Given the improvement, we believe the benefit of GUARDRAIL outweighs the runtime overhead.

Answer to RQ2. GUARDRAIL is effective in rectifying data errors and improving the accuracy of ML-integrated SQL queries with a modest runtime overhead.

Table 6: Runtime overheads (in seconds) and breakdown.

Dataset ID	#1	#2	#3	#4	#5	#6
GUARDRAIL Time	1.367	0.252	0.015	0.008	0.003	0.013
Inference Time	1.754	0.226	0.091	0.303	0.353	0.018
Dataset ID	#7	#8	#9	#10	#11	#12
GUARDRAIL Time	0.045	0.165	0.149	0.454	0.443	1.074
Inference Time	0.173	0.320	0.306	0.670	0.083	0.995

Table 7: Search space and enumeration time.

Dataset ID	#1	#2	#3	#4	#5	#6
# Attr.	15	5	40	9	10	4
# DAGs (w/ MEC)	216	1	5	8	5	8
Time (w/ MEC)	67	4	4	4	5	5
# DAGs (w/o MEC)	2.46×10^5	1.02×10^3	2.20×10^{13}	1.11×10^6	5.11×10^3	7.50×10^1
Dataset ID	#7	#8	#9	#10	#11	#12
# Attr.	28	7	21	17	31	18
# DAGs (w/ MEC)	8	120	18	60	168	180
Time (w/ MEC)	5	13	6	20	7	12
# DAGs (w/o MEC)	3.76×10^9	4.41×10^2	1.05×10^7	1.11×10^6	3.33×10^{10}	2.36×10^6

8.3 RQ3: Ablation Study and Impact Analysis

In this RQ, we study the usefulness of our MEC-based synthesis algorithm and auxiliary sampler as an ablation study. We also investigate the impact of the threshold ϵ on the coverage and loss of the synthesized integrity constraints.

To compare the effectiveness of the MEC-based synthesis algorithm, we first disable the MEC enumeration step in Alg. 2 and employ an `OptSMT` solver to directly optimize the program over the sketch by minimizing the loss (i.e., the number of violated examples). However, even on the smallest dataset with four attributes (# 4), the `OptSMT` solver runs out of time (i.e., 24 hours) without producing any *satisfiable* solution, letting alone the optimal one. For instance, the solver yields tens of millions clauses during the constraint solving and optimization process, which goes beyond

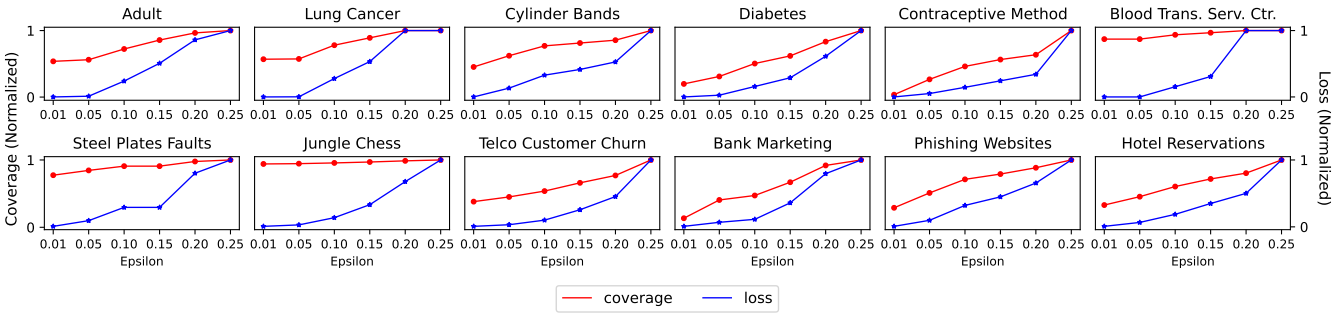

Figure 7: Impact of ϵ on coverage and loss.

Table 8: Effectiveness of the auxiliary sampler (measured by normalized coverage)

Dataset ID	#1	#2	#3	#4	#5	#6
w/o Auxiliary Sampler	0.393	0.623	0.179	0.000	0.000	0.000
w/ Auxiliary Sampler	0.395	0.741	0.442	0.126	0.109	0.394
Dataset ID	#7	#8	#9	#10	#11	#12
w/o Auxiliary Sampler	0.400	0.054	0.287	0.145	0.233	0.227
w/ Auxiliary Sampler	0.409	0.062	0.305	0.149	0.242	0.250

the capability of the solver. Since directly comparison is infeasible due to the scalability issue, we report the search space for both MEC-based and non-MEC-based enumeration in Table 7. We observe that, by leveraging the statistical inference to deduce the MEC, it significantly reduces the search space by order of magnitudes. For instance, the search space for dataset #3 is reduced from 2.20×10^{13} to 5 and the enumeration time on the reduced search space is generally manageable and moderate.

As expected, sampling from the auxiliary distribution leads to a higher coverage of the synthesized integrity constraints compared to using the original data (i.e., the identity sampler). As shown in Table 8, auxiliary sampler yields significantly better results (p-value $= 0.037 \leq 0.05$). More importantly, the identity sampler remains unusable on three datasets with the coverage of 0. We presume that it is because the high cardinality of the attributes in these datasets makes it difficult for off-the-shelf structure learning solutions to identify the PGM. In contrast, the auxiliary sampler is more robust and yields a higher coverage.

In Fig. 7, we investigate the impact of the threshold ϵ on the coverage of the synthesized integrity constraints. Here, ϵ controls tolerance of the integrity constraints to the noisy examples in the synthesis process. We observe that the coverage increases with the threshold ϵ for most datasets. As the cost of the coverage increases, the synthesized integrity constraints may become less accurate (i.e., higher loss; blue line in Fig. 7). Given the trade-off between the coverage and the loss, we recommend setting $\epsilon = 0.01 - 0.05$ for most datasets as it delivers a reasonable trade-off.

Answer to RQ3. Our MEC-based synthesis and the auxiliary sampler significantly boosts the effectiveness and efficiency of the synthesis process. The threshold ϵ plays a crucial role in balancing the coverage and the loss of the synthesized integrity constraints.

9 RELATED WORK

We have reviewed closely related work in Sec. 6. Here, we discuss the broader context of our work.

Data Quality Constraints. Data quality constraints are validation rules that define the acceptable values, relationships, and properties of the data. However, existing data validation solutions often focus on data errors such as numeric outliers [6, 22], mis-spellings [35], uniqueness constraints [7], value range constraints [31, 33], etc. Within these data quality frameworks, GUARDRAIL alleviates the burden of manual constraint specification by synthesizing parametric integrity constraints.

Program Synthesis in Database. Program synthesis aims to automatically generate a program that meets a given specification [14]. It has been used in many database applications, such as data transformation [11], SQL synthesis [34] and insight discovery [38]. Recently, program synthesis has been extended to generate programs from noisy examples [15–17]. However, as validated in our evaluation, traditional PBE methods (e.g., `OptSMT`-based synthesizer) are not scalable to handle the size of the datasets we consider.

ML Reliability. Our work is similar to the detection of concept drifts or outliers in ML inference [12, 18, 24, 41] and training data debugging [23, 37, 40, 42]. However, GUARDRAIL distinguishes itself in several aspects. Compared to the concept drift detection methods, GUARDRAIL is purely data-driven and does not rely on ML models while many existing works are dependent on the probability predicted by the ML models. As a constraint-based method, GUARDRAIL is naturally interpretable and can be easily integrated into the existing data processing pipeline (e.g., one can easily translate our DSL into standard SQL queries). And, compared to the training data debugging methods, GUARDRAIL primarily focuses on the data quality issues in the inference phase of the ML models.

10 CONCLUSION

In this paper, we propose a novel approach to solidify ML-integrated SQL queries with automated integrity constraint synthesis. We recast traditional integrity constraint discovery as a program synthesis problem from noisy and opaque input-output examples. We introduce a DSL and a sketch-based synthesis algorithm to facilitate program synthesis. Evaluations demonstrate the high effectiveness of our approach across various real-world datasets.

REFERENCES

[1] Extended version of the paper with appendix. https://anonymous.4open.science/r/prog-syn-126A/appendix.pdf, 2024.

[2] Research artifact. https://anonymous.4open.science/r/prog-syn-126A, 2024.

[3] Daniel W Barowy, Dimitar Gochev, and Emery D Berger. Checkcell: Data debugging for spreadsheets. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 507–523, 2014.

[4] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv:1708.03731v2 [stat.ML]*, 2019.

[5] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. *νz*-an optimizing smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 194–199. Springer, 2015.

[6] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[7] Tamraparni Dasu, Theodore Johnson, Shanmugauelayut Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 240–251, 2002.

[8] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[9] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering*, 23(5):683–698, 2010.

[10] Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. Conformance constraint discovery: Measuring trust in data-driven systems. In *Proceedings of the 2021 International Conference on Management of Data*, pages 499–512, 2021.

[11] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[13] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330, 2011.

[14] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

[15] Shivam Handa and Martin Rinard. Inductive program synthesis over noisy datasets using abstraction refinement based optimization. *arXiv preprint arXiv:2104.13315*, 2021.

[16] Shivam Handa and Martin Rinard. Program synthesis over noisy data with guarantees. *arXiv preprint arXiv:2103.05030*, 2021.

[17] Shivam Handa and Martin C Rinard. Inductive program synthesis over noisy data. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 87–98, 2020.

[18] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[19] Yka Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[20] Ihab F Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, 2004.

[21] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 683–698, 2017.

[22] Edwin M Knox and Raymond T Ng. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the international conference on very large data bases*, pages 392–403. Citeseer, 1998.

[23] Yanhui Li, Linghan Meng, Lin Chen, Li Yu, Di Wu, Yuming Zhou, and Baowen Xu. Training data debugging for the fairness of machine learning software. In *Proceedings of the 44th International Conference on Software Engineering*, pages 2215–2227, 2022.

[24] Haochuan Lu, Huanlin Xu, Nana Liu, Yangfan Zhou, and Xin Wang. Data sanity check for deep learning systems via learnt assertions. *arXiv preprint arXiv:1909.03835*, 2019.

[25] Christopher Meek. Strong completeness and faithfulness in bayesian networks. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 411–418, 1995.

[26] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. Preventing data errors with continuous testing. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, pages 373–384, 2015.

[27] Aditya V Nori, Sherjil Ozair, Sriram K Rajamani, and Deepak Vijaykeerthy. Efficient synthesis of probabilistic programs. *ACM SIGPLAN Notices*, 50(6):208–217, 2015.

[28] Orna Raz, Philip Koopman, and Mary Shaw. Semantic anomaly detection in online data sources. In *proceedings of the 24th International Conference on Software Engineering*, pages 302–312, 2002.

[29] Feras A Saad, Marco F Cusumano-Towner, Ulrich Schaechtle, Martin C Rinard, and Vikash K Mansinghka. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–32, 2019.

[30] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 281–294, 2005.

[31] Jie Song and Yeye He. Auto-validate: Unsupervised data validation using data-domain patterns inferred from data lakes. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1678–1691, 2021.

[32] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

[33] Dezhan Tu, Yeye He, Weiwei Cui, Song Ge, Haidong Zhang, Shi Han, Dongmei Zhang, and Surajit Chaudhuri. Auto-validate by-history: Auto-program data quality constraints to validate recurring data pipelines. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4991–5003, 2023.

[34] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466, 2017.

[35] Pei Wang and Yeye He. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 811–828, 2019.

[36] Marcel Wienöbst, Malte Luttermann, Max Bannach, and Maciej Liskiewicz. Efficient enumeration of markov equivalent dags. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12313–12320, 2023.

[37] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. Complaint-driven training data debugging for query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1317–1334, 2020.

[38] Junjie Xing, Xinyu Wang, and HV Jagadish. Data-driven insight synthesis for multi-dimensional data. *Proceedings of the VLDB Endowment*, 17(5):1007–1019, 2024.

[39] Jing Nathan Yan, Oliver Schulte, MoHan Zhang, Jiannan Wang, and Reynold Cheng. Scoded: Statistical constraint oriented data error detection. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 845–860, 2020.

[40] Jie Yang, Alisa Smirnova, Dingqi Yang, Gianluca Demartini, Yuan Lu, and Philippe Cudré-Mauroux. Scalpel-cd: leveraging crowdsourcing and deep probabilistic modeling for debugging noisy training data. In *The World Wide Web Conference*, pages 2158–2168, 2019.

[41] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2327–2344, 2021.

[42] Xuezhou Zhang, Xiaojin Zhu, and Stephen Wright. Training set debugging using trusted items. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[43] Yunjia Zhang, Zhihan Guo, and Theodoros Rekatsinas. A statistical perspective on discovering functional dependencies in noisy data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 861–876, 2020.

APPENDIX

A Proof of Proposition 1

Proof. We first introduce the faithfulness assumption and the Markov property.

Definition .1 (Faithfulness Assumption [32]). *A distribution P is faithful to a DAG G if and only if conditional independence relationships in P imply d-separations in G. Here, d-separation is a structural property of three sets of nodes in a DAG G. Two sets of nodes A and B are d-separated by a set of nodes C if and only if every path between a node in A and a node in B is blocked by C.*

Definition .2 (Markov Property [32]). *A distribution P is Markovian to a DAG G if and only if d-separations in G imply conditional independence relationships in P.*

The faithfulness assumption is a widely-used assumption and the de facto setup when learning PGMs. The main justification is that the set of unfaithful distributions is of measure zero (i.e., given a G, the set of such parameters has Lebesgue measure zero) [25]. Therefore, we take this assumption by default. Likewise, the Markov property is an inverse of the faithfulness assumption. The two assumptions forms a necessary and sufficient condition for the PGM to be a faithful representation of the underlying distribution. Now, we present our proof for Proposition 1.

By the definition of the SEM, since $a_{\boldsymbol{k}}$ are the parent variables of a_i, there exists direct edges from $a_{\boldsymbol{k}}$ to a_i, which implies that $a_i \not\perp\!\!\!\perp a_{\boldsymbol{k}}$; because there does not exist any other nodes that that block the path between a_i and $a_{\boldsymbol{k}}$ in G. □

B Proof of Theorem 4.1

Proof. Without loss of generality, let $s[\cdot]$ be an arbitrary statement sketch in $p[\cdot]$. In the context of the PGM, $s[\cdot]$ corresponds to one or a few directed edges from $a_{\boldsymbol{k}}$ to a_j in G, where $a_{\boldsymbol{k}}$ is the determinant attributes and a_j is the dependent attribute in $s[\cdot]$. By Proposition 1, $s[\cdot]$ is locally non-trivial. Therefore, we have that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$. To show that $s[\cdot]$ is globally non-trivial, we need to show that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$ on any subset of the data that is conditioned on other statement sketches in $p[\cdot]$ (i.e., D^b where b is a condition on other statement sketches in $p[\cdot]$). That is, the above condition is equivalent to showing that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid b$, where b is the condition. Since we are to show that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}}$ on any possible b, we can further rewrite the condition as $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid a_{\boldsymbol{z}}$ where $a_{\boldsymbol{z}}$ is any set of determinant attributes that are comprised by $p[\cdot]$. Since P_D is faithful to G, we have that $a_j \not\perp\!\!\!\perp a_{\boldsymbol{k}} \mid a_{\boldsymbol{z}}$ for any $a_{\boldsymbol{z}}$. Therefore, we have that $s[\cdot]$ is globally non-trivial. □

C Proof of Proposition 2

Proof. We first prove the presence of directed edges from each attribute in $a_{\boldsymbol{k}}$ to a_j in every DAG G within the MEC $[G]$ by contradiction. Assume that for some pair of attributes $a_l, a_m \in a_{\boldsymbol{k}}$, the triplet a_l, a_j, a_m does not form a v-structure in $[G]$. This means that we do not have the configuration $a_l \to a_j \leftarrow a_m$. According to the definition of faithfulness, the absence of a v-structure implies that for any set of attributes $a_{\boldsymbol{z}} \subset D$ that renders a_l and a_m conditionally independent, a_j must not be in $a_{\boldsymbol{z}}$. However, by the definition of a locally non-trivial statement sketch, we have that a_l is not conditionally independent of a_m given a_j, denoted as $a_l \not\perp\!\!\!\perp a_m \mid a_j$. This is a contradiction to the assumption that a_l, a_j, a_m do not form a v-structure, as the presence of a_j in the conditioning set should not affect the conditional independence of a_l and a_m if they do not form a v-structure. Therefore, the only consistent configuration under the MEC $[G]$ is that a_l, a_j, a_m must form a v-structure, which implies that there is a directed edge from a_l to a_j and from a_m to a_j in every DAG G within the MEC $[G]$.

Then, we prove the absence of other directed edges to a_j in every DAG G within the MEC $[G]$ by contradiction. Assume that there exists a directed edge from $a_q \notin a_{\boldsymbol{k}}$ to a_j in some DAG G within the MEC $[G]$. $a_q \to a_j \leftarrow a_m$ (where $a_m \in a_{\boldsymbol{k}}$) forms a v-structure in G. This implies that $a_q \not\perp\!\!\!\perp a_m \mid a_j$. However, by the definition of statement sketch, it is clear that $a_{\boldsymbol{k}}$ are the only determinant set of a_j. Therefore, the dependency between a_q and a_m must be mediated by a_j. That is, $a_q \perp\!\!\!\perp a_m \mid a_j$. This is a contradiction to $a_q \not\perp\!\!\!\perp a_m \mid a_j$. □

D Analysis of Auxiliary Distribution

Proposition 5. *Given a dataset D with joint probability distribution P_D for its attributes $a_1, \cdots, a_n$, and its auxiliary binary distribution $P_{\mathbb{I}}$ for the random vector $\mathbb{I}$, we have $a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}} \iff \mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}}$, where $\perp\!\!\!\perp$ denotes conditional independence, and $\boldsymbol{k}$ is a set of indices of attributes.*

Proof. To prove the proposition, we need to show that:

$$a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}} \quad \iff \quad \mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}},$$

where $\perp\!\!\!\perp$ denotes conditional independence.

We will prove both directions of the equivalence.

(1): Assume that $a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}}$. We will show that this implies $\mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}}$.

Let t_1 and t_2 be independent samples from P_D. Define the random variables:

$$\mathbb{I}_k = \begin{cases} 1 & \text{if } t_1(a_k) = t_2(a_k), \\ 0 & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{k}$ be a set of attribute indices, and consider $\mathbb{I}_{\boldsymbol{k}} = (\mathbb{I}_{k_1}, \ldots, \mathbb{I}_{k_m})$.

Given $\mathbb{I}_{\boldsymbol{k}} = \mathbf{1}$ (i.e., $\mathbb{I}_k = 1$ for all $k \in \boldsymbol{k}$), we have $t_1(a_{\boldsymbol{k}}) = t_2(a_{\boldsymbol{k}}) = \boldsymbol{x_k}$ for some value $\boldsymbol{x_k}$.

Since $a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}}$, we have for all x_i, x_j:

$$P(a_i = x_i, a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}) = P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}) \cdot P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}).$$

Because t_1 and t_2 are independent given $a_{\boldsymbol{k}} = \boldsymbol{x_k}$, the joint probability that $t_1(a_i) = t_2(a_i) = x_i$ and $t_1(a_j) = t_2(a_j) = x_j$ is:

$$\begin{aligned} &P(t_1(a_i) = x_i,\ t_2(a_i) = x_i,\ t_1(a_j) = x_j,\ t_2(a_j) = x_j \mid t_1(a_{\boldsymbol{k}}) = t_2(a_{\boldsymbol{k}}) = \boldsymbol{x_k}) \\ &\quad = [P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \cdot [P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2. \end{aligned}$$

Therefore, the probability that $\mathbb{I}_i = 1$ and $\mathbb{I}_j = 1$ given $\mathbb{I}_{\boldsymbol{k}} = \mathbf{1}$ is:

$$\begin{aligned} P(\mathbb{I}_i = 1, \mathbb{I}_j = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) &= \sum_{x_i, x_j} [P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \cdot [P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \\ &= \left(\sum_{x_i} [P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \right) \cdot \left(\sum_{x_j} [P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \right) \\ &= P(\mathbb{I}_i = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) \cdot P(\mathbb{I}_j = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}). \end{aligned}$$

Similarly, for $\mathbb{I}_i = b_i$ and $\mathbb{I}_j = b_j$ with $b_i, b_j \in \{0, 1\}$, we have:

$$P(\mathbb{I}_i = b_i, \mathbb{I}_j = b_j \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) = P(\mathbb{I}_i = b_i \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) \cdot P(\mathbb{I}_j = b_j \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}).$$

Thus, $\mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}$.

Next, consider $\mathbb{I}_{\boldsymbol{k}} = \boldsymbol{b_k}$ where at least one $b_k = 0$. In this case, $t_1(a_{\boldsymbol{k}}) \neq t_2(a_{\boldsymbol{k}})$ for some $k \in \boldsymbol{k}$. Since t_1 and t_2 are independent on the attributes where $t_1(a_k) \neq t_2(a_k)$, the variables $\mathbb{I}_i$ and $\mathbb{I}_j$ are conditionally independent given $\mathbb{I}_{\boldsymbol{k}} = \boldsymbol{b_k}$.

Therefore, in all cases, $\mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}}$.

(2): Assume that $\mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}}$. We will show that this implies $a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}}$.

Suppose, for contradiction, that a_i and a_j are not conditionally independent given $a_{\boldsymbol{k}}$. Then there exists $\boldsymbol{x_k}$, x_i, and x_j such that:

$$P(a_i = x_i, a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}) \neq P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}) \cdot P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k}).$$

Consider the probability that $\mathbb{I}_i = 1$ and $\mathbb{I}_j = 1$ given $\mathbb{I}_{\boldsymbol{k}} = \mathbf{1}$:

$$\begin{aligned} P(\mathbb{I}_i = 1, \mathbb{I}_j = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) &= \sum_{x_i, x_j} [P(a_i = x_i, a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \\ &\neq \left(\sum_{x_i} [P(a_i = x_i \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \right) \cdot \left(\sum_{x_j} [P(a_j = x_j \mid a_{\boldsymbol{k}} = \boldsymbol{x_k})]^2 \right) \\ &= P(\mathbb{I}_i = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}) \cdot P(\mathbb{I}_j = 1 \mid \mathbb{I}_{\boldsymbol{k}} = \mathbf{1}). \end{aligned}$$

This contradicts the assumption that $\mathbb{I}_i \perp\!\!\!\perp \mathbb{I}_j \mid \mathbb{I}_{\boldsymbol{k}}$. Therefore, $a_i \perp\!\!\!\perp a_j \mid a_{\boldsymbol{k}}$.

□

E Data Source

Table 9: Dataset description

Dataset Name	Link	Category	# Attributes	# Rows
Adult	https://archive.ics.uci.edu/dataset/2/adult	Demographic	15	48842
Lung Cancer	https://www.bnlearn.com/bnrepository/discrete-small.html#cancer	Medical	5	20000
Cylinder Bands	https://www.openml.org/search?type=data&id=6332	Manufacturing	40	540
Diabetes	https://www.openml.org/search?type=data&id=37	Medical	9	520
Contraceptive Method Choice	https://www.openml.org/search?type=data&id=23	Demographic	10	1473
Blood Transfusion Service Center	https://www.openml.org/search?type=data&id=1464	Medical	4	748
Steel Plates Faults	https://www.openml.org/search?type=data&id=40982	Manufacturing	28	1941
Jungle Chess	https://www.openml.org/search?type=data&id=41027	Game	7	44819
Telco Customer Churn	https://www.kaggle.com/datasets/blastchar/telco-customer-churn/data	Business	21	7043
Bank Marketing	https://archive.ics.uci.edu/dataset/222/bank+marketing	Business	17	45211
Phishing Websites	https://www.openml.org/search?type=data&id=4534	Security	31	11055
Hotel Reservations	https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset	Business	18	36275

Datasets. We describe the datasets used in the experiments in Table 9. The Adult and Bank Marketing datasets are two widely used datasets for classification tasks. The Lung Cancer dataset is a dataset used in the literature for causal structure learning tasks with a ground truth

causal graph, where some causal relationships enforces the integrity constraints on the data. The remaining datasets are from the OpenML repository or Kaggle and are used in the literature for tabular ML tasks.

F Case Study

To further demonstrate the effectiveness of GUARDRAIL, we conduct a case study on the Adult dataset to compare the results of ML-integrated SQL queries executed under three error-handling schemes: *ignore*, *rectify*, and *gt (ground truth)*.

The adult dataset contains demographic information of individuals, including their age, workclass, education. The task is to predict the income of an individual based on the demographic information. With GUARDRAIL, we can synthesize the integrity constraints from the data. Below, we show one of the synthesized integrity constraints and the rectification process to correct the prediction error.

```
...
GIVEN rel ON marital-status HAVING
  IF rel = Husband THEN marital-status ⟵ Married-civ-spouse;
  IF rel = Wife THEN marital-status ⟵ Married-civ-spouse
...
```
(9)

The above integrity constraint states that if the relationship (`rel`) is Husband or Wife, then the marital status should be Married-civ-spouse. Then, we launch a ML-integrated SQL query to calculate the average age of individuals from the private work class with different income.

```
SELECT adult.income_pred, AVG(adult.age) FROM adult GROUP BY adult.income_pred WHERE adult.workclass == 'Private';
```

We first execute the query on the clean data (ground truth).

	$\text{adult.income}_{\text{pred}}$	AVG(adult.age)
0	≤50K	34.90
1	>50K	43.72

However, for data with errors, there would result in a deviation from the ground truth, as shown below.

	$\text{adult.income}_{\text{pred}}$	AVG(adult.age)
0	≤50K	35.25 (0.35)
1	>50K	44.00 (0.28)

The error introduces a deviation of 0.35 and 0.28 in the mean age for the two income groups. Indeed, the deviation could attribute to the violation of the integrity constraint. For example, the following row violates the integrity constraint

ID	age	workclass	…	marital-status	…	$\text{income}_{\text{pred}}$
1064	45	Private	…	Separated	…	<=50K

Per the integrity constraint, given the individual's relationship is Husband, the marital status should be Married-civ-spouse instead of Separated. This error in turn leads to the prediction error in the income (from >50K to <=50K). With GUARDRAIL, we can rectify the error by correcting the marital status to Married-civ-spouse. By iterating the rectification process to all rows, we run the query again and obtain the corrected result.

	$\text{adult.income}_{\text{pred}}$	AVG(adult.age)
0	≤50K	34.90
1	>50K	43.72

As can be seen from the table above, the rectification process successfully corrects the prediction and reduces the deviation to 0.00 for both income groups.

Having said that, we acknowledge that in some cases, one row may be hard to correct due to the complex dependencies between the attributes. For instance, if the values for the marital status and relationship are both corrupted (e.g., the marital status is "Married-civ-spouse" → "Separated" and the relationship is "Husband" → "Not-in-family"), it is hard to determine the correct value for the marital status. In such cases, the rectification process may fail to correct the potential prediction error. However, we argue that such cases beyond the scope that the integrity constraints could cover, as the combination ("Separated" and "Not-in-family") stands for a valid relationship. In this case, the prediction error is not due to the violation of the integrity constraints, but rather the inherent ambiguity in the data.