ABSTRACT

As data is a central component of many modern systems, the cause of a system malfunction may reside in the data, and, specifically, particular properties of the data. For example, a health-monitoring system that is designed under the assumption that weight is reported in imperial units (lbs) will malfunction when encountering weight reported in metric units (kilograms). Similar to software debugging, which aims to find bugs in the mechanism (source code or runtime conditions), our goal is to debug the data to identify potential sources of disconnect between the assumptions about the data and the systems that operate on that data. Specifically, we seek which properties of the data cause a data-driven system to malfunction. We propose DataExposer, a framework to identify data properties, called profiles, that are the root causes of performance degradation or failure of a system that operates on the data. Such identification is necessary to repair the system and resolve the disconnect between data and system. Our technique is based on causal reasoning through interventions: when a system malfunctions for a dataset, DataExposer alters the data profiles and observes changes in the system’s behavior due to the alteration. Unlike statistical observational analysis that reports mere correlations, DataExposer reports causally verified root causes—in terms of data profiles—of the system malfunction. We empirically evaluate DataExposer on three real-world and several synthetic data-driven systems that fail on datasets due to a diverse set of reasons. In all cases, DataExposer identifies the root causes precisely while requiring orders of magnitude fewer interventions than prior techniques.

1 INTRODUCTION

Traditional software debugging aims to identify errors and bugs in the mechanism—such as source code, configuration files, and runtime conditions—that may cause a system to malfunction [24, 35, 49]. However, in modern systems, data has become a central component that itself can cause a system to fail. Data-driven systems comprise complex pipelines that rely on data to solve a target task. Prior work addressed the problem of debugging machine-learning models [12] and finding root causes of failures in computational pipelines [51], where certain values of the pipeline parameters—such as a specific model and/or a specific dataset—cause the pipeline failure. However, just knowing that a pipeline fails for a certain dataset is not enough; naturally, we ask: what properties of a dataset caused the failure?

Two common reasons for malfunctions in data-driven systems are: (1) incorrect data, and (2) disconnect between the assumptions about the data and the design of the system that operates on the data. Such disconnects may happen when the system is not robust, i.e., it makes strict assumptions about metadata (e.g., data format, domains, ranges, and distributions), and when new data drifts from the data over which the system was tested on before deployment [58] (e.g., when a system expects a data stream to have a weekly frequency, but the data provider suddenly switches to daily data).

Therefore, in light of a failure, one should investigate potential issues in the data. Some specific examples of commonly observed system malfunctions caused by data include: (1) decline of a machine-learned model’s accuracy (due to out-of-distribution data), (2) unfairness in model predictions (due to imbalanced training data), (3) excessive processing time (due to a system’s failure to scale to large data), and (4) system crash (due to invalid input combination in the data tuples beyond what the system was designed to handle). These examples indicate a common problem: disconnect or mismatch between the data and the system design. Once the mismatch is identified, then possible fixes could be either to repair the data to suit the system design, or to adjust the system design (e.g., modify source code) to accommodate data with different properties.

A naïve approach to deal with potential issues in the data is to identify outliers: report tuples as potentially problematic based on how atypical they are with respect to the rest of the tuples in the dataset. However, without verifying whether the outliers actually cause unexpected outcomes, we can never be certain about the actual root causes. As pointed out in prior work [7]: “With respect to a computation, whether an error is an outlier in the program’s input distribution is not necessarily relevant. Rather, potential errors can be spotted by their effect on a program’s output distribution.” To motivate our work, we start with an example taken from a real-world incident, where Amazon’s delivery service was found to be racist [43].

Example 1 (Biased Classifier). An e-commerce company wants to build an automated system that suggests who should get discounts. To this end, they collect information from the customers’ purchases over one year and build a dataset over the attributes name, gender, age, race, zip code, phone, products purchased, etc. Anita, a
A recent study of 112 high-severity incidents in Microsoft Azure captures the candidate root causes. For example, “outliers cause hypotheses” that are expressive enough to capture the candidate root causes. For example, “outliers cause unexpected outcomes” is just one of the many possible hypotheses, which offers very limited expressivity. Second, we need to verify the hypotheses to confirm or refute them, which enables us to pinpoint the actual root causes, eliminating false positives.

Data profile as root cause. Towards solving the first challenge, our observation is that data-driven systems often function properly for certain datasets, but malfunction for others. Such malfunction is often rooted in certain properties of the data, which we call data profiles, that distinguish passing and failing datasets. Examples include size of a dataset, domains and ranges of attribute values, correlations between attribute pairs, conditional independence [73], functional dependencies and their variants [14, 23, 40, 45, 54], and other more complex data profiles [18, 48, 55, 69].

Oracle-guided root cause identification. Our second observation is that if we have access to an oracle that can indicate whether the system functions desirably or not, we can verify our hypotheses. Access to an oracle allows us to precisely isolate the correct root causes of the undesirable malfunction from a set of candidate causes. Here, an oracle is a mechanism that can characterize whether the system functions properly over the input data. The definition of proper functioning is application-specific; for example, achieving a certain accuracy may indicate proper functioning for an ML pipeline. Such oracles are often available in many practical settings, and have been considered in prior work [24, 51].

Solution sketch. In this paper, we propose DataExposer, a framework that identifies and exposes data profiles that cause a data-driven system to malfunction. Our framework involves two main components: (1) an intervention-based mechanism that alters the profiles of a dataset, and (2) a mechanism that speeds up analysis by carefully selecting appropriate interventions. Given a scenario where a system malfunctions (fails) over a dataset but functions properly (passes) over another, DataExposer focuses on the discriminative profiles, i.e., data profiles that significantly differ between the two datasets. DataExposer’s intervention mechanism modifies the “failing” dataset to alter one of the discriminative profiles; it then observes whether this intervention causes the system to perform desirably, or the malfunction persists. DataExposer speeds up this analysis by favoring interventions on profiles that are deemed more likely causes of the malfunction. To estimate this likelihood, we leverage three properties of a profile: (1) coverage: the more tuples an intervention affects, the more likely it is to fix the system behavior; (2) discriminating power: the bigger the difference between the failing and the passing datasets over a profile, the more likely that the profile is a cause of the malfunction, and (3) attribute association: if a profile involves an attribute that is also involved with a large number of other discriminative profiles, then that profile has high likelihood to be a root cause. This is because altering such a profile is likely to passively repair other discriminative profiles as a side-effect (through the associated attribute). We also provide a group-testing-based technique that allows group intervention, which helps expedite the root-cause analysis further.

Scope. In this work, we assume knowledge of the classes of (domain-specific) data profiles that encompass the potential root causes. E.g., in Example 1, we assume the knowledge that correlation between attribute pairs and disparity between the conditional probability...
where no individual cell is significantly responsible, rather, a holistic approach: it removes one cell of the data at a time, and observes the effect. This approach leverages a few properties of the data profiles to efficiently explore the space of candidate root causes with a small number of interventions. Additionally, we develop an efficient group-testing-based algorithm that further reduces the number of required interventions. (Sec 4)

We evaluate DATAEXPOSER on three real-world applications, where data profiles are responsible for causing system malfunction, and demonstrate that DATAEXPOSER successfully explains the root causes with a very small number of interventions (< 5). Furthermore, DATAEXPOSER requires 10–1000× fewer interventions when compared against two state-of-the-art techniques for root-cause analysis: BugDoc [51] and Anchors [62]. Through an experiment over synthetic pipelines, we further show that the number of required interventions by DATAEXPOSER increases sub-linearly with the number of discriminative profiles, thanks to our group-testing-based approach. (Sec 5)

2 PRELIMINARIES & PROBLEM DEFINITION

In this section, we first formalize the notions of system malfunction and data profile, its violation, and transformation function used for intervention. We then proceed to define explanation (cause and corresponding fix) of system malfunction and formulate the problem of data-profile-centric explanation of system malfunction.

Basic notations. We use \( R(A_1, A_2, \ldots, A_m) \) to denote a relation schema over \( m \) attributes, where \( A_i \) denotes the \( i^{th} \) attribute. We use \( \text{Dom}_i \) to denote the domain of attribute \( A_i \). Then the set \( \text{Dom}^m = \text{Dom}_1 \times \cdots \times \text{Dom}_m \) specifies the domain of all possible tuples. A dataset \( D \subseteq \text{Dom}^m \) is a specific instance of the schema \( R \). We use \( t \in \text{Dom}^m \) to denote a tuple in the schema \( R \). We use \( t.A_i \in \text{Dom}_i \) to denote the value of the attribute \( A_i \) of the tuple \( t \) and use \( D.A_j \) to denote the multiset of values all tuples in \( D \) take for attribute \( A_j \).

2.1 Quantifying System Malfunction

To measure how much the system malfunctions over a dataset, we use the malfunction score.

Software testing and debugging techniques [3, 4, 16, 28, 32, 34, 38, 44, 49, 75] are either application-specific, require user-defined test suites, or rely only on observational data. The key contrast between software debugging and our setting is that the former focuses on white-box programs: interventions, runtime conditions, program invariants, control-flow graphs, etc., all revolve around program source code and execution traces. Unlike programs, where lines have logical and semantic connections, tuples in data do not have similar associations. Data profiles significantly differ in their semantics, and discovery and intervention techniques from program profiles, and, thus, techniques for program profiling do not trivially apply here. We treat data as a first-class citizen in computational pipelines, while considering the program as a black box.

Contributions. In this paper, we make the following contributions:

• We formalize the novel problem of identifying root causes (and fixes) of the disconnect between data and data-driven systems in terms of data profiles (and interventions). (Sec 2)

• We design a set of data profiles that are common root causes of data-driven system malfunctions, and discuss their discovery and intervention techniques based on available technology. (Sec 3)

• We design and develop a novel intervention approach to pinpoint causally verified root causes. The approach leverages a few properties of the data profiles to efficiently explore the space of candidate root causes with a small number of interventions. Additionally, we develop an efficient group-testing-based algorithm that further reduces the number of required interventions. (Sec 4)

• We evaluate DATAEXPOSER on three real-world applications, where data profiles are responsible for causing system malfunction, and demonstrate that DATAEXPOSER successfully explains the root causes with a very small number of interventions (< 5). Furthermore, DATAEXPOSER requires 10–1000× fewer interventions when compared against two state-of-the-art techniques for root-cause analysis: BugDoc [51] and Anchors [62]. Through an experiment over synthetic pipelines, we further show that the number of required interventions by DATAEXPOSER increases sub-linearly with the number of discriminative profiles, thanks to our group-testing-based approach. (Sec 5)
Definition 3 (Malfunction score). Let \( D \subseteq \text{Dom}\^m \) be a dataset, and \( S \) be a system operating on \( D \). The malfunction score \( m_S(D) \in [0, 1] \) is a real value that quantifies how much \( S \) malfunctions when operating on \( D \).

The malfunction score \( m_S(D) = 0 \) indicates that \( S \) functions properly over \( D \) and a higher value indicates a higher degree of malfunction, with 1 indicating extreme malfunction. A threshold parameter \( \tau \) defines the acceptable degree of malfunction and translates the continuous notion of malfunction to a Boolean value. If \( m_S(D) \leq \tau \), then \( D \) is considered to pass with respect to \( S \); otherwise, there exists a mismatch between \( D \) and \( S \), whose cause (and fix) we aim to expose.

Example 4. For a binary classifier, its misclassification rate (additive inverse of accuracy) over a dataset can be used as a malfunction score. Given a dataset \( D \), if a classifier \( S \) makes correct predictions for tuples in \( D' \subseteq D \), and incorrect predictions for the remaining tuples, then \( S \) achieves accuracy \( \frac{|D'|}{|D|} \), and, thus, \( m_S(D) = 1 - \frac{|D'|}{|D|} \).

Example 5. In fair classification, we can use disparate impact \([39]\), which is defined by the ratio between the number of tuples with favorable outcomes within the unprivileged and the privileged groups, to measure malfunction.

2.2 Profile-Violation-Transformation (PVT)

Once we detect existence of a mismatch, the next step is to investigate its cause. We characterize the issues in a dataset that are responsible for the mismatch between the dataset and the system using data profiles. Structure or schema of data profiles is given by profile templates, which contains holes for parameters. Parameterizing a profile template gives us a concretization of the corresponding profile (\( P \)). Given a dataset \( D \), we use existing data-profiling techniques to find out parameter values to obtain concretized data profiles, such that \( D \) satisfies the concretized profiles. To evaluate how much a dataset \( D \) satisfies or violates a data profile, we need a corresponding violation function (\( V \)). Violation functions provide semantics of the data profiles. Finally, to alter a dataset \( D \), with respect to a data profile and the corresponding violation function, we need a transformation function (\( T \)). Transformation functions provide the intervention mechanism to alter data profiles of a dataset and suggest fix to remove the cause of malfunction. DATAEXPOSER requires the following three components over the schema (Profile, Violation function, Transformation function), PVT in short:

1. \( P \): a (concretized) profile along with its parameters, which follows the schema (profile type, parameters).
2. \( V(D, P) \): a violation function that computes how much the dataset \( D \) violates the profile \( P \) and returns a violation score.
3. \( T(D, P, V) \): a transformation function that transforms the dataset \( D \) to another dataset \( D' \) such that \( D' \) no longer violates the profile \( P \) with respect to the violation function \( V \). (When clear from the context, we omit the parameters \( P \) and \( V \) when using the notation for transformation functions.)

For a PVT triplet \( X \), we define \( X_P \) as its profile, \( X_V \) as the violation function and \( X_T \) as the transformation function. We provide examples and additional discussions on data profiles, violation functions, and transformation functions in Section 3.

2.2.1 Data Profile. Intuitively, data profiles encode dataset characteristics. They can refer to a single attribute (e.g., mean of an attribute) or multiple attributes (e.g., correlation between a pair of attributes, functional dependencies, etc.).

Definition 6 (Data Profile). Given a dataset \( D \), a data profile \( P \) denotes properties or constraints that tuples in \( D \) (collectively) satisfy.

2.2.2 Profile Violation Function. To quantify the degree of violation a dataset incurs with respect to a data profile, we use a profile violation function that returns a numerical violation score.

Definition 7 (Profile violation function). Given a dataset \( D \) and a data profile \( P \), a profile violation function \( V(D, P) \mapsto [0, 1] \) returns a real value that quantifies how much \( D \) violates \( P \).

\( V(D, P) = 0 \) implies that \( D \) fully complies with \( P \) (does not violate it at all). In contrast, \( V(D, P) > 0 \) implies that \( D \) violates \( P \). The higher the value of \( V(D, P) \), the higher the profile violation.

2.2.3 Transformation Function. In our work, we assume knowledge of a passing dataset for which the system functions properly, and a failing dataset for which the system malfunctions. Our goal is to identify which profiles of the failing dataset caused the malfunction. We seek answer to the question: how to “fix” the issues within the failing dataset such that the system no longer malfunctions on it (mismatch is resolved)? To this end, we apply interventions causal reasoning: we intervene on the failing dataset by altering its attributes such that the profile of the altered dataset matches the corresponding correct profile of the passing dataset. To perform intervention, we need transformation functions with the property that it should push the failing dataset “closer” to the passing dataset in terms of the profile that we are interested to alter. More formally, after the transformation, the profile violation score should decrease.

Definition 8 (Transformation function). Given a dataset \( D \), a data profile \( P \), and a violation function \( V \), a transformation function \( T(D, P, V) \mapsto 2^{\text{Dom}\^m} \) alters \( D \) to produce \( D' \) such that \( V(D', P) = 0 \).

A dataset can be transformed by applying a series of transformation functions, for which we use the composition operator (\( \circ \)).

Definition 9 (Composition of transformations). Given a dataset \( D \), and two PVT triplets \( X \) and \( Y \), \( (X_P \circ Y_T)(D) = X_P(Y_T(D)) \). Further, if \( D'' = (X_T \circ Y_T)(D) \), then \( X_V(D'', X_P) = Y_V(D'', Y_P) = 0 \).

2.3 Problem Definition

We expose a set of PVT triplets for explaining the system malfunction. The explanation contains both the cause and the corresponding fix: profile within a PVT triplet indicates the cause of system malfunction with respect to the corresponding transformation function, which suggests the fix.

Definition 10 (Explanation of system malfunction). Given (1) a system \( S \) with a mechanism to compute \( m_S(D) \forall D \subseteq \text{Dom}\^m \), (2) an allowable malfunction threshold \( \tau \), (3) a passing dataset \( D_{\text{pass}} \) for which \( m_S(D_{\text{pass}}) \leq \tau \), (4) a failing dataset \( D_{\text{fail}} \) for which \( m_S(D_{\text{fail}}) > \tau \), and (5) a set of candidate PVT triplets \( X \) such that \( \forall X \in X \)

\( X_V(D_{\text{pass}}, X_P) = 0 \wedge X_V(D_{\text{fail}}, X_P) > 0 \),

the explanation of the malfunction of \( S \) for \( D_{\text{fail}} \), but not for \( D_{\text{pass}} \), is a set of PVT triplets \( X^* \subseteq X \) such that \( m_S((\forall X \in X^*, X_T)(D_{\text{fail}})) \leq \tau \).
Informally, $\mathcal{X}^*$ explains the cause: why $S$ malfunctions for $D_{\text{fail}}$ but not for $D_{\text{pass}}$. More specifically, failing to satisfy the profiles of the PVT triplets in $\mathcal{X}^*$ are the causes of malfunction. Furthermore, the transformation functions of the PVT triplets in $\mathcal{X}^*$ suggest the fix: how can we repair $D_{\text{fail}}$ to eliminate system malfunction. However, there could be many possible such $\mathcal{X}^*$ and we seek a minimal set $\mathcal{X}^*$ such that transformation for every $X \in \mathcal{X}^*$ is necessary to bring down the malfunction score below the threshold $\tau$.

**Definition 11 (Minimal explanation of system malfunction).** Given a system $S$ that malfunctions for $D_{\text{fail}}$ and an allowable malfunction threshold $\tau$, an explanation $\mathcal{X}^*$ of $S$’s malfunction for $D_{\text{fail}}$ is minimal if $\forall X' \subseteq \mathcal{X}^* \exists S((\forall X \in \mathcal{X}^*)\mathcal{X}_T(D_{\text{fail}})) > \tau$.

Note that there could be multiple such minimal explanations and we seek any one of them, as any minimal explanation exposes the causes of mismatch and suggests minimal fixes.

**Problem 12 (Discovering explanation of mismatch between data and system).** Given a system $S$ that malfunctions for $D_{\text{fail}}$ but functions properly for $D_{\text{pass}}$, the problem of discovering the explanation of mismatch between $D_{\text{fail}}$ and $S$ is to find a minimal explanation that captures (1) the cause why $S$ malfunctions for $D_{\text{fail}}$ but not for $D_{\text{pass}}$ and (2) how to repair $D_{\text{fail}}$ to remove the malfunction.

### Figure 1: A list of PVT triplets that we consider in this paper, their syntax, and semantics.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Data type</th>
<th>Discovery over $D$</th>
<th>Interpretation</th>
<th>Violation by $D$</th>
<th>Transformation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (DOMAIN, $A_j$, $S$)</td>
<td>Categorical</td>
<td>$S = {t.A_j}_{t \in D}$</td>
<td>Values are drawn from a specific domain.</td>
<td>$\sum_{t \in D}</td>
<td>f(A_j) \notin S</td>
</tr>
<tr>
<td>2 (DOMAIN, $A_j$, $S$)</td>
<td>Numerical</td>
<td>$S = [l, ub]$, where $l = \min_{t \in D} A_j$ and $ub = \max_{t \in D} A_j$</td>
<td>Values lie within a bound.</td>
<td>$\sum_{t \in D}</td>
<td>f(A_j) \notin S</td>
</tr>
<tr>
<td>3 (DOMAIN, $A_j$, $S$)</td>
<td>Text</td>
<td>$S = {t \in D</td>
<td>\pi \in P}$, where $\pi$ is a regex over $D.A_j$ learned via pattern discovery [56]</td>
<td>Values satisfy a regular expression or expression of values lie within a bound.</td>
<td>$\sum_{t \in D}</td>
</tr>
<tr>
<td>4 (OUTLIER, $A_j$, $O$, $\theta$)</td>
<td>All</td>
<td>$\theta = \frac{\sum_{t \in D}</td>
<td>O(D.A_j \mid A_j)| \theta}}{</td>
<td>|D</td>
<td></td>
</tr>
<tr>
<td>5 (MISSING, $A_j$, $\theta$)</td>
<td>All</td>
<td>$\theta = \frac{\sum_{t \in D}</td>
<td>f(A_j) \notin S</td>
<td>_{D}}{</td>
<td>|D</td>
</tr>
<tr>
<td>6 (SELECTIVITY, $P$, $\theta$)</td>
<td>All</td>
<td>$\theta = \frac{</td>
<td>|D</td>
<td></td>
<td>\mid f(D) \mid \theta)}{</td>
</tr>
<tr>
<td>7 (INDEP, $A_j$, $A_k$, $\alpha$)</td>
<td>Categorical</td>
<td>$\alpha$ denotes Chi-squared statistic between $D.A_j$ and $D.A_k$.</td>
<td>$\chi^2$ statistic between a pair of attributes is below a threshold with a p-value $\leq 0.05$.</td>
<td>$1 - e^{-\frac{\chi^2(D.A_j \mid D.A_k | \alpha)}{2}}$</td>
<td>Modify attribute values to remove/reduce dependence.</td>
</tr>
<tr>
<td>8 (INDEP, $A_j$, $A_k$, $\alpha$)</td>
<td>Numerical</td>
<td>$\alpha$ denotes Pearson correlation coefficient between $D.A_j$ and $D.A_k$.</td>
<td>PCC between a pair attributes is below a threshold with a p-value $\leq 0.05$.</td>
<td>$\frac{</td>
<td>|PCC(D.A_j \mid D.A_k | \alpha)}{1-\alpha}}$</td>
</tr>
<tr>
<td>9 (INDEP, $A_j$, $A_k$, $\alpha$)</td>
<td>Categorical, numerical</td>
<td>Learn causal graph and causal coefficients ($\alpha$) using TETRAD [66]</td>
<td>A causal relationship between a pair of attributes is unlikely (with causal coefficient less than $\alpha$).</td>
<td>$\max_\theta \left( \frac{</td>
<td>|f(D.A_j \mid D.A_k | \alpha)}{1-\alpha}} \right)$</td>
</tr>
</tbody>
</table>

### 3 DATA PROFILES, VIOLATION FUNCTIONS, & TRANSFORMATION FUNCTIONS

We now provide an overview of the data profiles we consider, how we discover them, how we compute the violation scores for a dataset w.r.t. a data profile, and how we apply transformation functions to alter profiles of a dataset. While a multitude of data-profiling primitives exist in the literature, we consider a carefully chosen subset of them that are particularly suitable for modeling issues in data that commonly cause malfunction or failure of a system. We focus on profiles that, by design, can better “discriminate” a pair of datasets as opposed to “generative” profiles (e.g., data distribution) that can profile the data better, but nonetheless are less useful for the task of discriminating between two datasets. However, the DataExposer framework is generic, and other profiles can be plugged into it.

As discussed in Section 2, a PVT triplet encapsulates a profile, and corresponding violation and transformation functions. Figure 1 provides a list of profiles along with the data types they support, how to learn their parameters from a given dataset, how to interpret them intuitively, and the corresponding violation and transformation functions. In this work, we assume that a profile can be associated with multiple transformation functions (e.g., rows 2 and 4), but each transformation function can be associated with at most one
profile. This assumption helps us to blame a unique profile as cause of the system malfunction when at least one of the transformation functions associated with that profile is verified to be a fix.

PVT triplets can be classified in different ways. Based on the strictness of the violation function, they can be classified as follows:

- **Strict**: All tuples are expected to satisfy the profile (rows 1–3).
- **Thresholded by data coverage**: Certain fraction $\theta$ of data tuples are allowed to violate the profile (rows 4–6).
- **Thresholded by a parameter**: Some degree of violation is allowed with respect to a specific parameter ($\alpha$) (rows 7–9).

Further, PVT triplets can be classified in two categories based on the nature of the transformation functions:

- **Local** transformation functions can transform a tuple in isolation without the knowledge of how other tuples are being transformed (e.g., rows 1–3). Some local transformation functions only transform the violating tuples (e.g., row 2, transformation (2)), while others transform all values (e.g., row 2, transformation (1)). For instance, in case of unit mismatch (kilograms vs. lbs), it is desirable to transform all values and not just the violating ones.
- **Global** transformation functions are holistic, as they need the knowledge of how other tuples are being transformed while transforming a tuple (e.g., rows 6 and 9).

**Example 13. Domain** requires two parameters: (1) an attribute $A_j \in R(D)$, and (2) a set $S$ specifying its domain. A dataset $D$ satisfies \( \langle \text{Domain}, A_j, S \rangle \) if $\forall t \in D \ t.A_j \in S$. The profile $\langle \text{Domain}, A_j, S \rangle$ is minimal w.r.t. $D$ if $\exists S' \subseteq S$ s.t. $D$ satisfies the profile $\langle \text{Domain}, A_j, S' \rangle$. The technique for discovering a domain $S$ varies depending on the data type of the attributes. Rows 1–3 of Figure 1 show three different domain-discovery techniques for different data types.

People\_fail (Figure 2) satisfies $\langle \text{Domain}, \text{gender}, \{ M, F \} \rangle$, as all tuples draw values from $\{ M, F \}$ for the attribute gender. Our case studies of Sentiment Prediction and Cardiovascular Disease Prediction show the application of the profile Domain (Section 5).

**Example 14. Outlier** requires three parameters: (1) an attribute $A_j \in R(D)$, (2) an outlier detection function $\text{O}(A, a) \Rightarrow \{ \text{True}, \text{False} \}$ that returns True if $a$ is an outlier w.r.t. the values within $A$, and False otherwise, and (3) a threshold $\theta \in [0, 1]$. A dataset $D$ satisfies $\langle \text{Outlier}, A_j, O, \theta \rangle$ if the fraction of outliers within the attribute $A_j$—according to $O$—does not exceed $\theta$. Otherwise, we compute how much the fraction of outliers exceeds the allowable fraction of outliers ($\theta$) and then normalize it by dividing by $1 - \theta$. The profile $\langle \text{Outlier}, A_j, O, \theta \rangle$ is minimal if $\exists \theta' < \theta$ s.t. $D$ satisfies $\langle \text{Outlier}, A_j, O, \theta' \rangle$.

An outlier detection function $O_{1.5}$ identifies values that are more than 1.5 standard deviation away from the mean as outliers. In People\_fail, age has a mean 34.5 and a standard deviation 11.78. According to $O_{1.5}$ only $1.5$—which is $0.1$ fraction of the tuples—is an outlier in terms of age as $t_5$’s age $60 > (34.5 + 1.5 \times 11.78) = 52.17$. Therefore, People\_fail satisfies $\langle \text{Outlier}, \text{age}, O_{1.5}, 0.1 \rangle$.

**Example 15. Indep** requires three parameters: two attributes $A_j, A_k \in R(D)$, and a real value $\alpha$. A dataset $D$ satisfies the profile $\langle \text{Indep}, A_j, A_k, \alpha \rangle$ if the dependency between $D.A_j$ and $D.A_k$ does not exceed $\alpha$. Different techniques exist to quantify the dependency and rows 6–9 of Figure 1 show three different ways to model dependency, where the first two are correlational and the last one is causal.

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>gender</th>
<th>age</th>
<th>race</th>
<th>zip code</th>
<th>phone</th>
<th>high expenditure</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>Shanice Johnson</td>
<td>F</td>
<td>45</td>
<td>A</td>
<td>01004</td>
<td>2085565977</td>
<td>no</td>
</tr>
<tr>
<td>t_2</td>
<td>DeShawn Bad</td>
<td>M</td>
<td>40</td>
<td>A</td>
<td>01004</td>
<td>2085374523</td>
<td>no</td>
</tr>
<tr>
<td>t_3</td>
<td>Malik Ayer</td>
<td>M</td>
<td>60</td>
<td>A</td>
<td>01005</td>
<td>2766865099</td>
<td>no</td>
</tr>
<tr>
<td>t_4</td>
<td>Dustin Jenifer</td>
<td>M</td>
<td>22</td>
<td>W</td>
<td>01009</td>
<td>7874891021</td>
<td>yes</td>
</tr>
<tr>
<td>t_5</td>
<td>Julietta Benn</td>
<td>F</td>
<td>41</td>
<td>W</td>
<td>01009</td>
<td>7872899033</td>
<td>yes</td>
</tr>
<tr>
<td>t_6</td>
<td>Molly Beadley</td>
<td>F</td>
<td>32</td>
<td>W</td>
<td>01004</td>
<td>2766865099</td>
<td>yes</td>
</tr>
<tr>
<td>t_7</td>
<td>Jake Bloom</td>
<td>M</td>
<td>25</td>
<td>W</td>
<td>01101</td>
<td>2084747803</td>
<td>yes</td>
</tr>
<tr>
<td>t_8</td>
<td>Lake Stonewald</td>
<td>M</td>
<td>35</td>
<td>W</td>
<td>01101</td>
<td>2042127741</td>
<td>yes</td>
</tr>
<tr>
<td>t_9</td>
<td>Scott Nossness</td>
<td>M</td>
<td>25</td>
<td>W</td>
<td>01101</td>
<td>2042127741</td>
<td>yes</td>
</tr>
<tr>
<td>t_{10}</td>
<td>Gabe Erwin</td>
<td>M</td>
<td>20</td>
<td>W</td>
<td>01102</td>
<td>4084241581</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Figure 2:** A sample dataset People\_fail with 10 entities. A logistic regression classifier trained over this dataset discriminates against African Americans (race = ‘A’) and women (gender = ‘F’) (Example 1).

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>gender</th>
<th>age</th>
<th>race</th>
<th>zip code</th>
<th>phone</th>
<th>high expenditure</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>Darin Brust</td>
<td>M</td>
<td>25</td>
<td>W</td>
<td>01004</td>
<td>2085565977</td>
<td>no</td>
</tr>
<tr>
<td>t_2</td>
<td>Rosalie Bad</td>
<td>F</td>
<td>22</td>
<td>W</td>
<td>01005</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>t_3</td>
<td>Kristine Hilyard</td>
<td>F</td>
<td>50</td>
<td>W</td>
<td>01004</td>
<td>2766865099</td>
<td>yes</td>
</tr>
<tr>
<td>t_4</td>
<td>Chase Ayer</td>
<td>F</td>
<td>22</td>
<td>A</td>
<td>787491021</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>t_5</td>
<td>Julietta Mchugh</td>
<td>F</td>
<td>51</td>
<td>W</td>
<td>01009</td>
<td>9042899033</td>
<td>yes</td>
</tr>
<tr>
<td>t_6</td>
<td>Doria Ely</td>
<td>F</td>
<td>32</td>
<td>W</td>
<td>01101</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>t_7</td>
<td>Kristan Whidden</td>
<td>F</td>
<td>25</td>
<td>W</td>
<td>01101</td>
<td>4047747803</td>
<td>no</td>
</tr>
<tr>
<td>t_8</td>
<td>Rene Strelow</td>
<td>M</td>
<td>35</td>
<td>W</td>
<td>01101</td>
<td>6362127741</td>
<td>yes</td>
</tr>
<tr>
<td>t_9</td>
<td>Ariel Brent</td>
<td>M</td>
<td>45</td>
<td>W</td>
<td>01102</td>
<td>4089065769</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Figure 3:** A sample dataset People\_pass with 9 entities. A logistic regression classifier trained over this dataset does not discriminate against any specific race or gender, and, thus, is fair (Example 1).

\((\text{Indep}, \text{race}, \text{high_expenditure}, 0.67)\) is satisfied by People\_fail using the PVT triplet of row 7, as $x^2$-statistic between race and high_expenditure over People\_fail is 0.67. We show the application of the profile Indep in our case study involving the task of Income Prediction in Section 5.

While the profiles in Figure 1 are defined over the entire data, analogous to conditional functional dependency [23], an extension to consider is conditional profiles, where only a subset of the data is required to satisfy the profiles.

### 4 INTERVENTION ALGORITHMS

We now describe our intervention algorithms to explain the mismatch between a dataset and a system malfunctioning on that dataset. Our algorithms consider a failing and a passing dataset as input and report a collection of PVT triplets (or simply PVTs) as the explanation (cause and fix) of the observed mismatch. To this end, we first identify a set of discriminative PVTs—whose profiles take different values in the failing and passing datasets—as potential explanation units, and then intervene on the failing dataset to alter the profiles and observe change in system malfunction. We develop two approaches that differ in terms of the number of PVTs considered simultaneously during an intervention. DATAEXPOSER\_GRID is a greedy approach that considers only one PVT at a time. However, in worst case, the number of interventions required by DATAEXPOSER\_GRID is linear in number of discriminative PVTs. Therefore, we propose a second algorithm DATAEXPOSER\_GRIST, built in the group-testing paradigm, that considers multiple PVTs to reduce the number of interventions, where the number of required interventions is logarithmically dependent on the number of discriminative PVTs. We start with an example scenario to demonstrate how DATAEXPOSER\_GRID works and then proceed to describe our algorithms.
4.1 Example Scenario

Consider the task of predicting the attribute high_expenditure to determine if a customer should get a discount (Example 1). The system calculates bias of the trained classifier against the unprivileged groups (measured using disparate impact [39]) as its malfunction score. We seek the causes of mismatch between this prediction pipeline and People_fail (Figure 2), for which the pipeline fails with a malfunction score of 0.75. We assume the knowledge of People_pass (Figure 3), for which the malfunction score is 0.15. The goal is to identify a minimal set of PVTs whose transformation functions bring down the malfunction score of People_fail below 0.20.

(Step 1) The first goal is to identify the profiles whose parameters differ between People_fail and People_pass. To do so, DataExposerGRID identifies the exhaustive set of PVTs over People_pass and People_fail and discards the identical ones (PVTs with identical profile-parameter values). We call the PVTs of the passing dataset whose profile-parameter values differ over the failing dataset discriminative PVTs. Figure 5 lists a few profiles of the discriminative PVTs w.r.t. People_pass and People_fail.

(Step 2) Next, DataExposerGRID ranks the set of discriminative PVTs based on their likelihood of offering an explanation of the malfunction. Our intuition here is that if an attribute A is related to the malfunction, then many PVTs containing A in their profiles would differ between People_fail and People_pass. Additionally, altering A with respect to one PVT is likely to automatically “fix” other PVTs associated with A.1 Based on this intuition, DataExposerGRID constructs a bipartite graph, called PVT-attributed graph, with discriminative PVTs on one side and data attributes on the other side (Figure 4). In this graph, a PVT X is connected to an attribute A if X_P is defined over A. In the bipartite graph, the degree of an attribute A captures the number of discriminative PVTs associated with A. During intervention, DataExposerGRID prioritizes PVTs associated with a high-degree attributes. For instance, since high_expenditure has the highest degree in Figure 4, PVTs associated with it are considered for intervention before others.

(Step 3) DataExposerGRID further ranks the subset of the discriminative PVTs that are connected to the highest-degree attributes in the PVT-attributed graph based on their benefit score. Benefit score of a PVT X encodes the likelihood of reducing system malfunction when the failing dataset is altered using X_T. The benefit score of X is estimated from (1) the violation score that the failing dataset incurs w.r.t. X_T, and (2) the number of tuples in the failing dataset that are altered by X_T. For example, to break the dependence between high_expenditure and race, the transformation corresponding to Indep modifies five tuples in People_fail by perturbing (adding noise to) high_expenditure. In contrast, the transformation for Missing needs to change only one value (t_6 or t_10). Since more tuples are affected by the former, it has higher likelihood of reducing the malfunction score. The intuition behind this is that if a transformation alters more tuples in the failing dataset, the more likely it is to reduce the malfunction score. This holds particularly in applications where the system optimizes aggregated statistics such as accuracy, recall, F-score, etc.

Figure 4: PVT-attribute graph. The attribute high_expenditure is associated with two discriminative PVTs. For ease of exposition, we only show profile within a PVT to denote the entire PVT.

Figure 5: A list of PVTs that discriminate People_pass (Figure 3) and People_fail (Figure 2) based on the scenario of Example 1. We omit the violation and transformation functions for ease of exposition.

(Step 4) DataExposerGRID starts intervening on People_fail using the transformation of the PVT corresponding to the profile (Indep, race, high_expenditure, 0.04) as its transformation offers the most likely fix. Then, it evaluates the malfunction of the system over the altered version of People_fail. Breaking the dependence between high_expenditure and race helps reduce bias in the trained classifier, and, thus, we observe a malfunction score of 0.35 w.r.t. the altered dataset. This exposes the first explanation of malfunction.

(Step 5) DataExposerGRID then removes the processed PVT (Indep) from the PVT-attributed graph, updates the graph according to the altered dataset, and re-iterates steps 2–4. Now the PVT corresponding to the profile Selectivity is considered for intervention as it has the highest benefit score. To do so, DataExposerGRID oversamples tuples corresponding to female customers with high_expenditure = yes. This time, DataExposerGRID intervenes on the transformed dataset obtained from the previous step. After this transformation, bias of the learned classifier further reduces and the malfunction score falls below the required threshold. Therefore, with these two interventions, DataExposerGRID is able to expose two issues that caused undesirable behavior of the prediction model trained on People_fail.

(Step 6) DataExposerGRID identifies an initial explanation over two PVTs: Indep and Selectivity. However, to verify whether it is a minimal, DataExposerGRID tries to drop from it one PVT at a time to obtain a proper subset of the initial explanation that is also an explanation. This procedure guarantees that the explanation only consists of PVTs that are necessary, and, thus, is minimal. In this case, both Indep and Selectivity are necessary, and, thus, are part of the minimal explanation. DataExposerGRID finally reports the following as a minimal explanation of the malfunction, where failure to satisfy the profiles is the cause and the transformations indicate fix (violation and transformation functions are omitted).

\[
\{ (\text{Indep}, \text{race}, \text{high}_{-}\text{expenditure} = 0.04),
(\text{Selectivity}, \text{gender} = \text{F} \land \text{high}_{-}\text{expenditure} = \text{yes}, 0.44) \}\]
4.2 Assumptions and Observations

We now proceed to describe our intervention algorithms more formally. We first state our assumptions and then proceed to present our observations that lead to the development of our algorithms.

Assumptions. DataExposer makes the following assumptions:

(A1) The ground-truth explanation of malfunction is captured by at least one of the discriminative PVTs. This assumption is prevalent in software-debugging literature where program predicates are assumed to be expressive enough to capture the root causes [24, 49].

(A2) If the fix corresponds to a composition of transformations, then the malfunction score achieved after applying the composition of transformations is less than the malfunction score achieved after applying any of the constituents, and all these scores are less than the transformation score achieved after applying the dataset PVTs to the original dataset.

We make the following observations:

(O1) If the ground-truth explanation of malfunction corresponds to a PVT, then multiple PVTs that involve the same attribute are likely to differ across the passing and failing datasets. This observation motivates us to prioritize interventions based on PVTs that are associated with high-degree attributes in the PVT-attribute graph. Additionally, intervening on the data based on one such PVT is likely to result in an automatic “fix” of other PVTs connecting via the high-degree attribute. For example, adding noise to high expenditure in Example 1 breaks its dependence with not only race but also with other attributes.

(O2) The PVT for which the failing dataset incurs a higher violation score is more likely to be a potential explanation of malfunction.

(O3) A transformation function that affects a large number of tuples in the dataset (O3). Formally, the benefit score is more likely to be a potential explanation of malfunction.

Algorithm 1: DataExposer (greedy)

Input: Failing dataset $D_{fail}$, passing dataset $D_{pass}$, malfunction score threshold $\tau$

Output: A minimal explanation set of PVTs $X^*$

1. $X_p \leftarrow \text{Discover-PVT}(D_{fail})$
2. $X_f \leftarrow \text{Discover-PVT}(D_{pass})$
3. $X_0^* \leftarrow X_f \cap X_p$ /* Common PVTs*/
4. $X \leftarrow X_f \setminus X_0^*$ /* Discriminative PVTs*/
5. $GP_A(\langle V_G, E_G \rangle) \leftarrow \text{Construct-PVT-Attr-Graph}(X, D_{fail})$
6. $B \leftarrow \text{Calculate-Benefit-Score}(X, GP_A, D_{fail})$
7. $X^* \leftarrow \emptyset$ /* Initialize minimal explanation set to be empty*/
8. $D \leftarrow D_{fail}$ /* Initialize dataset to the failing dataset*/
9. while $m_S(D) > \tau$ do
10. $X_{hda} = \{X | (X, A) \in E_G \land A = \max_{A \in \mathcal{R}(D)} \deg_G(A)\}$ /* PVTs adjacent to high-degree attributes in $GP_A$*/
11. $X = \arg \max_{X \in X_{hda}} B(X)$ /* Highest-benefit PVT*/
12. $\Delta = m_S(D) - m_S(X_f(D))$ /* Malfunction reduction*/
13. $GP_A \leftarrow GP_A \setminus \{X\}$ /* Update $GP_A$*/
14. if $\Delta > 0$ then /* Reduces malfunction*/
15. $D \leftarrow X_f(D)$ /* Apply transformation*/
16. $GP_A \setminus \{X\}$ /* Update the PVT-attribute graph*/
17. $B \leftarrow GP_A \setminus \{X\}$ /* Update benefit scores*/
18. $X^* \leftarrow X^* \cup \{X\}$ /* Add P to explanation set*/
19. $X \leftarrow X \setminus \{X\}$ /* Remove P from the candidates*/
20. $X^* = \text{Make-Minimal}(X^*)$ /* Obtain minimality of $X^*$*/
21. return $X^*$ /* $X^*$ is a minimal explanation*/

PVT-attribute graph. DataExposer leverages observation O1 by constructing a bipartite graph (GP_A), called PVT-attribute graph, with all attributes $A \in \mathcal{R}(D)$ as nodes on one side and all discriminative PVTs $X \in X$ on the other side. An attribute $A$ is connected to a PVT $X$ if and only if $X_f$ has $A$ as one of its parameters. E.g., Figure 4 shows the PVT-attribute graph w.r.t. People fail and People pass (Example 1). In this graph, the PVT corresponding to (Indep, race, high expenditure) is connected to two attributes, race and high expenditure. Intuitively, this graph captures the dependence relationship between PVTs and attributes, where an intervention with respect to a PVT $X$ modifies an attribute $A$ connected to it. If this intervention reduces the malfunction score then it could possibly fix other PVTs that are connected to $A$.

Benefit score calculation. DataExposer uses the aforementioned observations to compute a benefit score for each PVT to model their likelihood of reducing system malfunction if the corresponding transformation is used to modify the failing dataset $D_{fail}$. Intuitively, it assigns a high score to a PVT with a high violation score (O2) and if the corresponding transformation function modifies a large number of tuples in the dataset (O3). Formally, the benefit score of a PVT $X$ is defined as the product of violation score of $D_{fail}$ w.r.t. $X_f$ and the “coverage” of $X_f$. The coverage of $X_f$ is defined as the fraction of tuples that it modifies. Note that the benefit calculation procedure acts as a proxy of the likelihood of a PVT to offer an explanation, without actually applying any intervention.

4.3 Greedy Approach

Algorithm 1 presents the pseudocode of our greedy technique DataExposer, which takes a passing dataset $D_{pass}$ and a failing dataset $D_{fail}$ as input and returns the set of PVTs that corresponds to a minimal explanation of system malfunction.

Lines 1-2 Identify two sets of PVTs $X_f$ and $X_p$ satisfied by $D_{fail}$ and $D_{pass}$, respectively.

Lines 3-4 Discard the PVTs $X_f \cap X_p$ from $X_p$ and consider the remaining discriminative ones $X \equiv X_p \setminus X_f$ as candidates for potential explanation of system malfunction.

Line 5 Compute the PVT-attribute graph $GP_A$, where the candidate PVTs $X$ correspond to nodes on one side and the data attributes correspond to nodes on the other side.

Line 6 Calculate the benefit score of each discriminative PVT $X \in X$ w.r.t. $D_{fail}$. This procedure relies on the violation score using the violation function of the PVT and the coverage of the corresponding transformation function over $D_{fail}$.
We now present our second algorithm (when the number of discriminative PVTs is odd, then the size of intervention) to evaluate the change in malfunction score. If a partition \( X \) is transformed according to the transformation \( D \), then the set of discriminative PVTs into two “almost” equal subsets \( X = X^* \). If the transformed dataset incurs a violation score less than \( \tau \) then \( X^* \) is replaced with \( X' \).

4.4 Group-testing-based Approach

We now present our second algorithm \( \text{DataExposer}_{GT} \), which performs group interventions to identify the minimal explanation that exposes the mismatch between a dataset and a system. The group intervention methodology is applicable under the following assumptions along with assumptions \( A_1 \) and \( A_2 \) (Section 4.2).

A3) The malfunction score incurred after applying a composition of transformations is less than the malfunction score incurred by the original dataset if and only if at least one of the constituent transformations reduces the malfunction score. For two PVTs \( X \) and \( Y \), if \( s_s((X \cup Y)_{D_{fail}}) < s_s(D_{fail}) \), then \( s_s((X \cup Y)_{D_{fail}}) < s_s(D_{fail}) \) or \( s_s((X \cup Y)_{D_{fail}}) < s_s(D_{fail}) \). Note that this assumption is crucial to consider group interventions and is prevalent in the group-testing literature [20].

\( \text{DataExposer}_{GT} \) follows the classical adaptive group testing (GT) paradigm [20] for interventions. To this end, it iteratively partitions the set of discriminative PVTs into two “almost” equal subsets (when the number of discriminative PVTs is odd, the size of the two partitions will differ by one). During each iteration, all PVTs in a partition are considered for intervention together (group intervention) to evaluate the change in malfunction score. If a partition does not help reduce the malfunction score, all PVTs within that partition are discarded. While traditional GT techniques [20] would use a random partitioning of the PVTs, \( \text{DataExposer}_{GT} \) can leverage the dependencies among PVTs (inferred from the PVT-attribute graph) to achieve more effective partitioning. Intuitively, it is beneficial to assign all PVTs whose transformations operate on the same attribute to the same partition, which is likely to enable aggressive pruning of spurious PVTs that do not reduce malfunction.

**Algorithm 2: DataExposer_{GT}** (group-testing-based)

**Input:** Failing dataset \( D_{fail} \), passing dataset \( D_{pass} \), malfunction score threshold \( \tau \)

**Output:** A minimal explanation set of PVTs \( X^* \)

```
1. \( X_f \leftarrow \text{Discover-PVT}(D_{fail}) \)
2. \( X_p \leftarrow \text{Discover-PVT}(D_{pass}) \)
3. \( X_p \leftarrow X_f \cap X_p \) /* Common PVTs */
4. \( X \leftarrow X_p \setminus X_f \) /* Discriminative PVTs */
5. \( G_{PA}(V_{G}, E_{G}) \leftarrow \text{Construct-PVT-attr-Graph}(X, D_{fail}) \)
6. \( D, X^* \leftarrow \text{Group-Test}(X, D_{fail}, G^2_{PA}, \tau) \) /* Obtain an exp. */
7. \( X^* \leftarrow \text{Make-Minimal}(X^*) \) /* Obtain minimality of \( X^* \) */
8. return \( X^* \) /* \( X^* \) is a minimal cause */
```

\( \text{DataExposer}_{GRD} \) captures the dependencies among PVTs by constructing a PVT-dependency graph \( G_{PD} \). Two PVTs \( U \) and \( V \) are connected by an edge in \( G_{PD} \) if they are connected via some attribute in \( G_{PA} \). \( G_{PD} \) is equivalent to \( G_{PA} \) (transitive closure of \( G_{PA} \)), restricted to PVT nodes (excluding the attribute nodes). This ensures that PVTs that are associated via some attribute in \( G_{PA} \) are connected in \( G_{PD} \). \( \text{DataExposer}_{GRD} \) partitions \( G_{PD} \) such that the number of edges between PVTs in different partitions is minimized. More formally, we aim to construct two “almost” equal-sized partitions of \( X \) such that the number of edges between PVTs from different partitions are minimized, which maps to the problem of finding the minimum bisection of a graph [31]. The minimum bisection problem is NP-hard [31] and approximate algorithms exist [26, 27]. In this work, we use the local search algorithm [26] (details are in our technical report [29]).

We proceed to demonstrate the benefit of using \( \text{DataExposer}_{GT} \) as opposed to traditional GT with the following example.

**Example 16.** Consider a set of 8 PVTs \( X = \{X_1, \ldots, X_8\} \) where the ground-truth (minimal) explanation is either \( \{X_1, X_8\} \) or \( \{X_4, X_8\} \) (disjunction). An example of steps for a traditional adaptive GT approach is shown in Figure 6(c). In this case, it requires a total of 14 interventions. Note that adaptive GT is a randomized algorithm and this example demonstrates one such execution. However, we observed similar results for other instances. In contrast to adaptive GT, \( \text{DataExposer}_{GT} \) constructs a min-bisection of the graph during each iteration: it does not partition \( \{X_2, X_3\} \) and \( \{X_5, X_7\} \) as none of these PVTs help reduce the malfunction. Therefore, it requires only 10 interventions.

Algorithm 2 presents the \( \text{DataExposer}_{GT} \) algorithm. It starts with a set of discriminative PVTs \( X \) and the PVT-attribute graph \( G_{PA} \). All candidate PVTs are then considered by \( \text{Group-Test} \) subroutine to identify the explanation \( X^* \).

**Group-Test.** Algorithm 3 presents the procedure that takes the set of discriminative PVTs \( X \), a failing dataset \( D \), PVT-dependency graph \( G_{PD} \), and the malfunction score threshold \( \tau \) as input. It returns a transformed (fixed) dataset and an explanation.

```
1. Initialize the solution set \( X^* \) to \( \emptyset \).
2-3. Return the candidate PVT set \( X \) if its cardinality is 1.
4. Partition \( X \) into \( X_1 \) and \( X_2 \) using min-bisection of the PVT-dependency graph \( G_{PD} \).
5. Calculate the malfunction score of the input dataset.
6. Calculate the reduction in malfunction score \( \Delta \) if \( D \) is intervened w.r.t. all PVTs \( X_1 \).
```

9
Algorithm 3: GROUP-TEST

Input: Candidate PVT $X$, dataset $D$, PVT-dependency graph $G_{PD}$, malfunction score threshold $\tau$

Output: A transformed dataset $D'$ and an explanation set of PVTs $X^*$

1. $X^* \leftarrow \emptyset$  
   /* Initialize explanation set to be empty */
2. if $|X| = 1$ then  
   /* Only a single PVT is candidate */
   3. return $T_X(D), X$
4. $X_1, X' \leftarrow \text{GET-MIN-BISECTION}(G_{PD}, X)$  
   /* Partition $X$ */
5. $M \leftarrow m_S(D)$  
   /* Initial malfunction score */
6. $\Delta_1 \leftarrow M - m_S(X_1^2(D))$  
   /* Malfunction reduction by $X_1^2$ */
7. if $M - \Delta_1 > \tau$ then  
   /* $X_1$ alone is insufficient */
   8. $\Delta_2 \leftarrow M - m_S(X_2^2(D))$  
   /* Malfunction reduction by $X_2^2$ */
9. if ($M - \Delta_1 \leq \tau$ OR ($\Delta_1 > 0$ AND $M - \Delta_2 > \tau$)) then  
   /* $X_1$, is sufficient OR $X_1$ helps AND $X_2$ is insufficient */
   10. $D, X' \leftarrow \text{GROUP-TEST}(X_1, D, G_{PD})$
   11. $X^* = X' \cup X^*$  
   /* Augment explanation set */
   12. if $M - \Delta_1 \leq \tau$ then  
   /* Malfunction is acceptable */
   13. return $D, X^*$
   14. if $\Delta_2 > 0$ then  
   /* $G_{PD}$ reduces malfunction */
   15. $D, X' \leftarrow \text{GROUP-TEST}(X_2, D, G_{PD})$
   16. $X^* = X' \cup X^*$  
   /* Augment explanation set */
17. return $D, X^*$

Lines 7-8 If the malfunction exceeds $\tau$ even after intervening on $D$ w.r.t. all PVTs in $X_1$ then try out $X_2$: calculate the reduction in malfunction score $\Delta_2$ if $D$ is intervened w.r.t. all PVTs in $X_2$.

Lines 9-13 Recursively call GROUP-TEST on the partition $X_1$ if one of the following conditions hold: (1) Intervening on $D$ w.r.t. all PVTs in $X_1$ reduces the malfunction to be lower than $\tau$: the explanation over $X_1$ is returned as the final explanation. (2) Intervening on $D$ w.r.t. all PVTs in $X_1$ reduces the malfunction, but still remains above $\tau$, but intervening on $D$ w.r.t. all PVTs in $X_2$ brings the malfunction below $\tau$: the explanation returned by the recursive call on $X_2$ is added to the set $X^*$ and $X_2$ is processed next.

Lines 14-16 Recursively call GROUP-TEST on $X_2$ if intervening on all PVTs in $X_2$ reduces malfunction. The set of PVTs returned by this recursive call of the algorithm are added to the solution set $X^*$.

Discussion on DATAEXPOSER_{GRD} vs. DATAEXPOSER_{GT}, DATAEXPOSER_{GRD} intervenes by considering a single discriminative PVT at a time. Hence, in the worst case, it requires $O(\log |X|)$ interventions where $X$ denotes the set of discriminative PVTs. Note that DATAEXPOSER_{GRD} requires much fewer interventions in practice and would require $O(|X|)$ only when any of the mentioned observations (O1–O3) do not hold. In contrast, DATAEXPOSER_{GT} performs group intervention by recursively partitioning the set of discriminative PVTs. Thus, the maximum number of interventions required by DATAEXPOSER_{GT} is $O(t \log |X|)$ where $t$ denotes the number of PVTs that help reduce malfunction if the corresponding profile is altered. Note that, in expectation, DATAEXPOSER_{GT} requires fewer interventions than DATAEXPOSER_{GRD} whenever $t = o(\log |X|/\log |X|)$. DATAEXPOSER_{GT} is particularly helpful when multiple PVTs disjunctively explain the malfunction. However, DATAEXPOSER_{GT} requires an additional assumption assumption A3 (Section 4.4). We discuss the empirical impact of this assumption in Section 5.1 (Cardiovascular disease prediction). Overall, we conclude that DATAEXPOSER_{GT} is beneficial for applications whenever $t = O(|X|/\log |X|)$ and observations O1-O3 hold (more details are in our technical report [29]).

5 EXPERIMENTAL EVALUATION

Our experiments involving DATAEXPOSER aim to answer the following questions: (Q1) Can DATAEXPOSER correctly identify the cause and corresponding fix of mismatch between a system and a dataset for which the system fails? (Q2) How efficient is DATAEXPOSER compared to other alternative techniques? (Q3) Is DATAEXPOSER scalable with respect to the number of discriminating PVTs?

Baselines. Since there is no prior work on modifying a dataset according to a PVT, we adapted state-of-the-art debugging and explanation techniques to incorporate profile transformations and explain the cause of system failure. We consider three baselines: (1) BugDoc [51] is a recent debugging technique that explores different parameter configurations of the system to understand its behavior. We adapt BugDoc to consider each PVT as a parameter of the system and interventions as the modified configurations of the pipeline. (2) Anchor [62] is a local explanation technique for classifiers that explains individual predictions based on a surrogate model. We train Anchor with PVTs as features, and the prediction variable is Pass/Fail where Pass (Fail) denotes the case where an input dataset incurs malfunction below (above) $\tau$. In this technique, each intervention creates a new data point to train the surrogate model. (3) GrpTest [20] is an adaptive group testing approach that performs group interventions to expose the mismatch between the input dataset and the system. It is similar to DATAEXPOSER_{GT} with a difference that the recursive partitioning of PVTs is performed randomly without exploiting the PVT-dependency graph.

5.1 Real-world Case Studies

We design three case studies focusing on three different applications, where we use well-known ML models [1, 2, 57] as black-box systems. For all of the three case studies, we use real-world datasets. Figure 7 presents a summary of our evaluation results.
Sentiment Prediction. The system in this study predicts sentiment of input text (reviews/tweets) and computes mismatch classification rate as the malfunction score. It uses flair [2], a pre-trained classifier to predict sentiment of the input records and assumes a target attribute in the input data, indicating the ground truth sentiment: A value of 1 for the attribute target indicates positive sentiment and a value of −1 indicates negative sentiment. We test the system over IMDb dataset [41] (50K tuples) and a twitter dataset [68] (around 1.6M tuples). The malfunction score of the system on the IMDb dataset is only 0.09 while on the twitter dataset it is 1.0. We considered IMDb as the passing dataset and twitter as the failing dataset and used both DataExposerGRD and DataExposerGT to explain the mismatch between the twitter dataset and the system. The ground-truth cause of the malfunction is that the target attribute in the twitter dataset uses “4” to denote positive and “0” to denote negative sentiment [68]. DataExposerGRD identifies a total of 3 discriminative PVTs between the two datasets. One such PVT includes the profile Domain of the target attribute that has corresponding parameter \( S = \{-1, 1\} \) for IMDb and \( S = \{0, 4\} \) for the twitter dataset. DataExposerGRD performs two interventions and identifies that the malfunction score reduces to 0.36 by mapping \( 0 \rightarrow -1 \) and \( 4 \rightarrow 1 \) by intervening w.r.t. the PVT corresponding Domain, which is returned as an explanation of the malfunction.

DataExposerGT and GrpTest both require 3 group interventions to explain the cause of system malfunction. BugDoc and Anchor require 10 and 303 interventions, respectively. Anchor calculates system malfunction on datasets transformed according to various local perturbations of the PVTs in the failing dataset.

Income Prediction. The system in this study trains a Random Forest classifier [57] to predict the income of individuals while ensuring fairness towards marginalized groups. The pipeline returns the normalized disparate impact [39] of the trained classifier w.r.t. the protected attribute (sex), as the malfunction score. Our input data includes census records [21] containing demographic attributes of individuals along with information about income, education, etc. We create two datasets through a random selection of records, and manually add noise to one of them to break the dependence between target and sex. The system has malfunction score of 0.195 for the passing dataset and 0.58 for the failing dataset due to the dependence between target and sex. DataExposerGRD identifies a total of 43 discriminative PVTs and constructs a PVT-attribute graph. In this graph, the target attribute has degree 15 while all other attributes have degree 2. The PVTs that include target are then intervened in non-increasing order of benefit. The transformation w.r.t. INDEPENDENT PVT on the target attribute breaks the dependence between target and all other attributes, thereby reducing the malfunction score to 0.32. Therefore, DataExposerGRD requires one intervention to explain the cause of the malfunction.

Our group testing algorithm DataExposerGT and GrpTest require 8 and 10 interventions, respectively. Note that group testing is not very useful because the datasets contain few discriminative PVTs.

BugDoc and Anchor do not identify discriminative PVTs explicitly and consider all PVTs (136 for this dataset) as candidates for intervention. Anchor performs 800 local interventions to explain the malfunction. BugDoc identifies the ground truth malfunction in 50% of the runs when allowed to run fewer than 10 interventions. It identifies the mismatch with intervention budget of 20 but the returned solution of PVTs is not minimal. For instance, BugDoc returns two PVTs: \( \{\text{INDEPENDENT}, \text{target}, \text{education}\} \) and \( \{\text{INDEPENDENT}, \text{target}, \text{sex}\} \) as the explanation of malfunction.

Cardiovascular Disease Prediction. This system trains an AdaBoost classifier [1] on patients’ medical records [13] containing age, height (in centimeters) and weight along with other attributes. It predicts if the patient has a disease and does not optimize for false positives. Therefore, the system calculates recall over the patients having cardiovascular disease, and the goal is to achieve more than 0.70 recall. The pipeline returns the additive inverse of recall as the malfunction score. We tested the pipeline with two datasets generated through a random selection of records: (1) the passing dataset satisfies the format assumptions of the pipeline; (2) for the failing dataset we manually converted height to inches. DataExposerGRD identifies 86 discriminative PVTs with height, weight and age having the highest degree of 15 in the PVT-attribute graph. Among the PVTs involving these attributes, the Domain of height has the maximum benefit, which is the ground-truth PVT too. DataExposerGRD alters the failing dataset by applying a linear transformation and it reduces the malfunction from 0.71 to 0.30. This explanation matches the ground truth difference in the passing and the failing dataset. Among baselines, BugDoc and Anchor performed 100 and 5900 interventions, respectively. Group testing techniques are not applicable because assumption A3 (Section 4.4) does not hold. We observe that the malfunction score with a composition of transformation functions is higher than the one in the original dataset if the composition involves INDEPENDENT PVT. This behavior is observed because adding noise to intervene with respect to INDEPENDENT PVT worsens the classifier performance. If we remove PVTs that violate this assumption, then DataExposerGT and GrpTest require 6 and 9 interventions, respectively.

Efficiency. Figure 7 presents the execution time of considered techniques for real-world applications presented above. DataExposerGRD, DataExposerGT and GrpTest are highly efficient and require less than 30 seconds to explain the ground-truth cause of malfunction. In contrast, Anchor is extremely inefficient as it requires more than 143 minutes for cardiovascular, while DataExposerGRD and BugDoc explain the malfunction within 63 seconds.

Key takeaways. Among all real-world case studies, the greedy approach DataExposerGRD requires the fewest interventions to explain the cause of malfunction. Group testing techniques, DataExposerGT and GrpTest, require fewer interventions than BugDoc and Anchor whenever assumption A3 (Section 4.4) holds. Anchor requires the highest number of interventions as it performs many
local transformations to explain the cause of failure. BugDoc optimizes interventions by leveraging combinatorial design algorithms: it requires more interventions than DataExposer but fewer than Anchor.

5.2 Synthetic Pipelines
We evaluate the effectiveness and scalability of DataExposerGRD and DataExposerGT for a diverse set of synthetic scenarios.

DataExposerGRD vs. DataExposerGT. In this experiment, we consider a pipeline where the ground-truth explanation of malfunction violates the observations discussed in Section 4.2. Specifically, the explanation requires modifying one particular value in the dataset and its likelihood (as estimated by DataExposerGRD) is ranked 54 among the set of discriminative PVTs. Therefore, it requires 54 interventions to explain the cause of malfunction. On the other hand, DataExposerGT performs group interventions and requires only 9 interventions. This experiment demonstrates that DataExposerGT requires fewer interventions than DataExposerGRD when the failing dataset and the corresponding PVTs do not satisfy the observations DataExposerGRD relies on. We present additional experiments with complex conjunctive and disjunctive explanations of malfunction in our technical report [29].

 Scalability. To test the scalability of our techniques, we compare their running time with increasing number of attributes and discriminative PVTs. Figure 8 shows that the time required by DataExposerGRD and DataExposerGT to explain the malfunction grows sub-linearly in the number of attributes and discriminative PVTs. We observe similar trend of the number of required interventions on varying these parameters. This experiment demonstrates that DataExposerGRD requires fewer than $O(|X|)$ interventions in practice (where $X$ denotes the set of discriminative profiles) and validates the logarithmic dependence of DataExposerGT on $|X|$.

6 RELATED WORK
Interventional debugging. AID [24] uses an interventional approach to blame runtime conditions of a program for causing failure; but it is limited to software bugs and does not intervene on datasets. BugDoc [51] finds parameter settings in a black-box pipeline as root causes of pipeline failure; but it only reports whether a dataset is a root cause and does not explain why a dataset causes the failure. CADET [42] uses causal inference to derive root causes of non-functional faults for hardware platforms focused on performance issues. Capuchin [65] casts fairness in machine learning as a database repair problem and adds or removes rows in the training data to simulate a fair world; but it does not aim to find cause of unfairness.

Data explanation. Explanations for query results have been abundantly studied [5, 17, 22, 71]. Some works find causes of errors in data generation processes [71], while others discover relationships among attributes [5, 22], and across datasets [17]. ExceLint [6] exploits the spatial structure of spreadsheets to look for erroneous formulas. Unlike interventional efforts, these approaches operate on observational data, and do not generate additional test cases.

Model explanation. Machine learning interpreters [61, 62] perturb testing data to learn a surrogate for models. Their goal is not to find mismatch between data and models. Debugging methods for

ML pipelines are similar to data explanation [11, 12], where training data may cause model’s underperformance. [70] and [46] discuss principled ways to find reasons of malfunctions. Wu et al. [72] allow users to complain about outputs of SQL queries, and presents data points whose removal resolves the complaints. [67] validates when models fail on certain datasets and assumes knowledge of the mechanism that corrupts the data. We aim to find discriminative profiles among datasets without such knowledge.

Causal inference debugging. Data-driven approaches have been taken for causal-inference-based fault localization [3, 4, 16, 34], software testing [28, 32, 38, 44, 74], and statistical debugging [49, 75]. However, they use a white-box strategy or are application-specific. Causal relational learning [64] infers causal relationships in relational data, but it does not seek mismatches between the data and the systems. Our work shares similarity with BugEx [63], which generates test cases to isolate root causes. However, it assumes complete knowledge of the program, and data-flow paths.

Data debugging. Porting concepts of debugging from software to data has gained attention in data management community [10, 53]. Dagger [59, 60] provides data debugging primitives for white-box interactions with data-driven pipelines. CheckCell [7] ranks data cells that unusually affect output of a given target. However, it is not meant for large datasets where single cells are unlikely to causes malfunction. Moreover, CheckCell cannot expose combination of root causes. DataExposer is general-purpose, application-agnostic, and interventional, providing causally verified issues mismatch between the data and the systems.

7 SUMMARY AND FUTURE DIRECTIONS
We introduced the problem of identifying causes and fixes of mismatch between data and systems that operate on data. To this end, we presented DataExposer, a framework that reports violation of data profiles as causally verified root causes of system malfunction and reports fixes in the form of transformation functions. We demonstrated the effectiveness and efficacy of DataExposer in explaining the reason of mismatch in several real-world and synthetic data-driven pipelines, significantly outperforming the state of the art. In future, we want to extend DataExposer to support more complex classes of data profiles. Additionally, we plan to investigate ways that can facilitate automatic repair of both data and the system guided by the identified data issues.


Sentiment 140 dataset. https://www.kaggle.com/kazanova/sentiment140


Sentiment 140 dataset. https://www.kaggle.com/kazanova/sentiment140


