Stress-Testing ML Pipelines with Adversarial Data Corruption

Jiongli Zhu
University of California, San Diego
USA
jiz143@ucsd.edu

Geyang Xu
University of California, San Diego
USA
gexu@ucsd.edu

Felipe Lorenzi
University of California, San Diego
USA
florenzi@ucsd.edu

Boris Glavic
University of Illinois Chicago
USA
bglavic@uic.edu

Babak Salimi
University of California, San Diego
USA
bsalimi@ucsd.edu

ABSTRACT

Structured data-quality issues—such as missing values correlated with demographics, culturally biased labels, or systemic selection biases—routinely degrade the reliability of machine-learning pipelines. Regulators now increasingly demand evidence that high-stakes systems can withstand these realistic, interdependent errors, yet current robustness evaluations typically use random or overly simplistic corruptions, leaving worst-case scenarios unexplored.

We introduce Savage, a causally inspired framework that (i) formally models realistic data-quality issues through dependency graphs and flexible corruption templates, and (ii) systematically discovers corruption patterns that maximally degrade a target performance metric. Savage employs a bi-level optimization approach to efficiently identify vulnerable data subpopulations and fine-tune corruption severity, treating the full ML pipeline, including preprocessing and potentially non-differentiable models, as a black box. Extensive experiments across multiple datasets and ML tasks (data cleaning, fairness-aware learning, uncertainty quantification) demonstrate that even a small fraction (around 5%) of structured corruptions identified by Savage severely impacts model performance, far exceeding random or manually crafted errors, and invalidating core assumptions of existing techniques. Thus, Savage provides a practical tool for rigorous pipeline stress-testing, a benchmark for evaluating robustness methods, and actionable guidance for designing more resilient data workflows.

PVLDB Reference Format:
Jiongli Zhu, Geyang Xu, Felipe Lorenzi, Boris Glavic, and Babak Salimi. Stress-Testing ML Pipelines with Adversarial Data Corruption. PVLDB, 18(11): 4668 - 4681, 2025.
doi:10.14778/3749646.3749721

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/lodino/savage.

1 INTRODUCTION

Machine-learning pipelines now approve loans, trigger sepsis alerts, and guide parole decisions—roles critical enough that policymakers increasingly demand reliability under "reasonably foreseeable" failures. For instance, Article 15 of the EU Artificial Intelligence Act mandates that high-risk AI systems achieve and maintain appropriate accuracy, robustness, and resilience throughout their lifecycle, while the NIST AI Risk-Management Framework explicitly calls for managing harmful bias and data-quality faults [1, 36]. Meeting these mandates is challenging because real-world tabular data rarely contain tidy, independent errors. Missing values, label flips, and selection biases typically arise through *structured, interdependent* processes: a "low-risk" flag in medical records suppresses lab tests while correlating with insurance status and demographics [13, 15]; loan datasets omit repayment histories precisely for subpopulations most likely to default [10, 33, 39]; and crowdsourced labels drift with cultural nuances, producing systematic misannotations [17]. Figure 1 confirms how pervasive such structure is—in a large public census corpus, the missingness of values are highly predictable from other attributes. These overlapping errors quietly erode accuracy, fairness, and generalizability [16], yet without provenance metadata, practitioners have no principled way to certify robustness against such realistic errors.

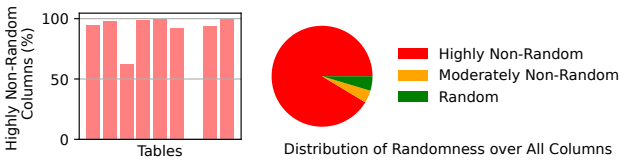


Figure 1: We analyzed 398 public census tables (>1,100 columns containing gaps). Treating each column's missingness as a binary label and predicting it from other attributes, we found that 91% of columns achieved an F1 score above 0.9 (bars). This indicates that missing values are predominantly systematic rather than random.

To rigorously address this critical issue, this paper introduces Savage (**S**ensitivity **A**nalysis **V**ia **A**utomatic **G**eneration of **E**rrors), a framework that systematically and automatically generates *realistic, high-impact, and adversarial* data corruption scenarios to stress-test end-to-end ML pipelines. Unlike existing benchmarks and adversarial attacks, Savage identifies corruption patterns that mirror plausible real-world conditions, for instance, revealing how non-random missingness, label errors, and selection bias jointly exacerbate model failure in vulnerable subpopulations. By automatically discovering these complex yet *interpretable* worst-case scenarios, Savage helps practitioners and researchers uncover critical pipeline vulnerabilities and develop demonstrably more robust, fair, and trustworthy ML systems.

Existing robustness benchmarks, including REIN [3], JENGA [46], CleanML [28], and Shades-of-Null [23], typically inject simplistic faults such as uniformly missing values, random label flips, or narrow demographic filters [16, 19]. As summarized in Table 1, these approaches do not systematically explore realistic, structured, and

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749721

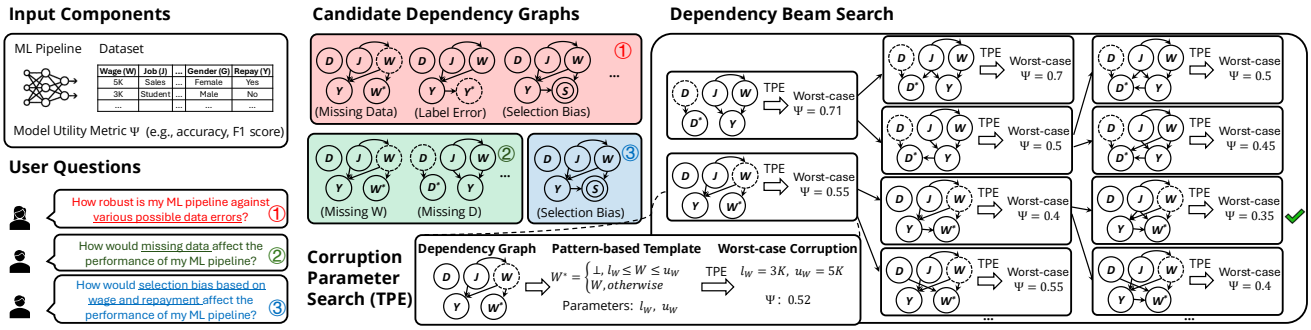


Figure 2: Overview of SAVAGE. Different user questions map to varying constraints on the dependency graphs, where ① denotes no constraints, ② restricts to missing data, and ③ specifies one unique dependency graph for selection bias. The dependency beam search is conducted to search the dependency graph and its corresponding worst-case concrete corruption (obtained by TPE) that leads to the lowest model utility measured by Ψ.

Paper	Data Corruption	Targeted Errors	Adversarial Analysis
Budach et al. [34]	✓	✗	✗
Islam et al. [19]	✓	✓	✗
Guha et al. [16]	✗	-	-
CleanML [28]	✓	✗	✗
REIN [3]	✓	✗	✗
JENGA [46]	✓	✓	✗
Shades-of-Null [23]	✓	✓	✗
SAVAGE (ours)	✓	✓	✓

Table 1: Comparison of existing works on data corruption, targeted errors, and adversarial analysis.

interdependent error patterns, thus underestimating the severity of data-quality issues in high-stakes applications [3, 16, 28]. Indiscriminate data-poisoning attacks also manipulate training data, but they generally target a specific model rather than evaluating entire ML pipelines. Moreover, most poisoning methods, including subpopulation and clean-label attacks [20, 31, 32, 35], are gradient-based, which craft subtle, low-level perturbations that require white-box gradient access, rendering them ineffective when pipelines include non-differentiable preprocessing or must be treated as black boxes.

By contrast, SAVAGE produces *explicit, interpretable* corruption patterns—such as missingness or label errors tied to specific demographic or semantic attributes—that mirror interpretable, real-world data-quality issues extensively adopted in ML and data-management studies [20, 30, 40, 43, 45]. By treating the entire ML pipeline, including cleaning, feature engineering, and model training, as a black box, SAVAGE uncovers worst-case yet plausible failure scenarios that neither existing benchmarks nor model-specific poisoning attacks can reveal.

At the heart of SAVAGE is a principled, *causally inspired* framework for modeling realistic data errors that arise from interdependent mechanisms (Section 3). These mechanisms are captured in a directed *dependency graph*: nodes represent clean attributes and their corrupted counterparts, while an edge means that the *probability or severity of error* in a child node depends on the *value* of its parent, e.g., if insurance-status is "self-pay," the wage field is more likely to be recorded as missing. The graph does not encode causal relations among clean attributes; it records only the pathways by which errors propagate. Dependency graphs can be seeded from domain knowledge or discovered automatically with a beam search (Figure 2). Each graph is paired with flexible, pattern-based *corruption templates* that specify *when* and *how* an attribute is corrupted, such as missingness triggered by demographics or label flips tied to textual cues. User-defined plausibility constraints (historical frequency thresholds, regulatory rules, validation checks) prune unrealistic scenarios. Together, dependency graphs and corruption templates form an interpretable *Data Corruption Process* that systematically captures common data-quality issues, including missing values, label errors, and selection bias.

Building on this formal foundation, we formulate the discovery of worst-case corruption scenarios as a *bi-level optimization* problem (Section 4). As shown in Figure 2, an upper-level combinatorial beam search [49] probes the space of all possible error dependencies, e.g., which attributes influence corrupted attributes, that maximally degrades a target metric such as accuracy or fairness. The lower-level Bayesian optimization (BO) tunes parameters to maximize performance degradation for a given dependency graph. Crucially, this approach treats the entire ML pipeline, including preprocessing and training steps, as a black-box function, requiring no gradient information. To further improve scalability, SAVAGE uses a proxy-based strategy: it efficiently identifies harmful corruption patterns using a computationally inexpensive proxy pipeline and transfers these patterns to resource-intensive ML frameworks, achieving substantial runtime improvements (over 10× speed-up on datasets of millions of rows).

To assess SAVAGE in practice, we conduct extensive experiments across multiple datasets and ML tasks (Section 5), thoroughly evaluating the impact of systematically generated corruptions on (i) data-cleaning and preparation methods [12, 24, 25, 27, 38, 44], (ii) fair and robust learning approaches [21, 41, 42, 53], and (iii) uncertainty quantification techniques [26, 52]. Our results demonstrate that even small, structured corruptions identified by SAVAGE can invalidate key assumptions about missingness or label stability, resulting in severe performance degradation that substantially exceeds the impact of random or manually crafted errors. In summary, our contributions include: (1) a unified, mechanism-aware Data Corruption Process for modeling realistic data-quality errors; (2) a gradient-free, interpretable bi-level optimization method to systematically identify adverse data corruptions; and (3) extensive empirical evidence

demonstrating significant, previously unrecognized vulnerabilities in state-of-the-art ML pipelines.

2 RELATED WORK

Benchmarks for Data Quality in machine learning (ML). As shown in Table 1, most prior work introduces synthetic errors or considers limited corruption scenarios without comprehensive analysis. Guha et al.[16] evaluate automated data cleaning but assume identical error distributions in both training and test data, thus failing to capture distribution shifts. JENGA[46] examines test-time corruptions while keeping training data clean, missing the impact of biased training data. CleanML[28], REIN[3], and Budach et al.[34] study data corruption effects but do not focus on specific error types or detrimental failure modes. Islam et al.[19] and Shades-of-Null[23] incorporate targeted corruption mechanisms, yet none perform systematic adversarial analysis to stress-test ML pipelines. Savage fills these gaps by generating structured, realistic data corruptions that reveal failure modes across data cleaning, fair learning, and uncertainty quantification (UQ). Unlike previous efforts, Savage systematically explores error-generation mechanisms and simulates worst-case corruption scenarios to uncover vulnerabilities that remain hidden under standard benchmark conditions. By integrating adversarial analysis, Savage enables a more rigorous evaluation of ML robustness under structured, non-random corruptions, offering a more realistic assessment of pipeline reliability.

Data Poisoning. Our work also relates to data poisoning, which deliberately alters training data to degrade model performance, misclassify specific examples, or implant backdoors [5, 6, 8, 11, 20, 31, 32, 47, 48, 50].Poisoning methods can be *targeted*, aiming to mislabel particular test instances [11, 50], or *indiscriminate*, broadly reducing overall accuracy [20, 31]. Our work shares a high-level goal with indiscriminate data poisoning: both aim to identify corruptions that degrade model performance. However, the motivations, constraints, and techniques differ significantly. Poisoning attacks typically seek to evade detection and therefore impose imperceptibility constraints on the perturbations, without requiring the modifications to reflect realistic data quality issues. In contrast, our objective is to evaluate the robustness of ML pipelines with realistic and often systematic data errors. Unlike poisoning approaches, Savage explicitly models structured corruptions such as selection bias, label errors, and missing values. In addition, our method accommodates both cases where users have limited knowledge of potential data issues and cases where domain-specific error types are known and can be specified.

Another key distinction is that Savage operates on full ML pipelines, including non-differentiable components like imputation or outlier removal. Most poisoning techniques either assume access to a differentiable end-to-end model or ignore preprocessing altogether. While some recent efforts [29] incorporate preprocessing (e.g., feature selection), they target narrow scenarios and do not generalize to broader pipeline components.

3 MODELING DATA CORRUPTION

We now introduce a principled framework for simulating the mechanisms by which real-world data collection processes generate corrupted datasets. Inspired by structural causal modeling [37, 54], this framework explicitly models how systematic dependencies between attributes, noise, and selection processes lead to data-quality issues such as missingness, label errors, and selection bias. Formally, let a dataset D consist of N tuples and have n attributes $\mathbf{A} = \{A_1, \ldots, A_n\}$. Given a tuple t, we use $t[A]$ to denote its value in attribute A. For any set of attributes $\mathbf{A}$, we let $\mathrm{Dom}(\mathbf{A})$ denote their joint domain, and $\mathbf{x}$ a specific assignment in that domain. The domains of all attributes are assumed to contain the special value $\perp$ that is used to mark missing values. The goal of this framework is to model the generation of a corrupted dataset $\tilde{D}$ from D by specifying mechanisms that govern how attributes are altered or omitted through noise-driven, interdependent processes. We use A_i to denote an attribute in the clean dataset and A_i^* to denote the corresponding corrupted attribute.

A data corruption process (DCP) consists of a *dependency graph* that models at the schema level which attributes of the clean dataset D and noise variables modelling stoachstic factors determine the values of the corrupted version A^* of an attribute A. Specifically, for each tuple t in the clean dataset, its corrupted counterpart t^* is created by computing the value of each corrupted attribute A^* based on its parents in the dependency graph using pattern-based corruption functions to be introduced in the following. Noise variables are used to model randomness in the corruption process. For instance, consider a simple label flipping example for a binary label attribute Y where the corrupted label Y^* has a certain probability to being flipped. That is, the corrupted label is computed solely based on the original label and the value of a noise variable N, resulting in the dependency structure: $N \rightarrow Y^* \leftarrow Y$. As another example, consider that for a subpopulation with specific demographics (attribute D), the corrupdated (binary) label is always 0, and otherwise the corrupted label is equal to the original label: $D \rightarrow Y^* \leftarrow Y$.

Definition 1 (Dependency Graph). *A* dependency graph $G = (\mathcal{V}, \mathcal{E})$ *is a directed acyclic graph whose vertex set* $\mathcal{V}$ *includes:*

- Original attributes, $\mathbf{A} = \{A_1, \ldots, A_n\}$,
- Corrupted counterparts, $\mathbf{A}^* = \{A_1^*, \ldots, A_n^*\}$, *where each* A_i^* *replaces* A_i *in the corrupted dataset,*
- Noise variables, $\mathbf{N} = \{N_1, \ldots, N_m\}$
- *An optional* binary selection indicator, *S, modeling data exclusion.*

Each directed edge $(u, v) \in \mathcal{E}$ *specifies that node* u *directly influences node* v. *For a corrupted attribute* $A^* \in \mathbf{A}^*$, *its* parent set $\boldsymbol{Pa}(A^*) \subseteq \mathcal{V}$ *designates all the variables that govern its corrupted value. We require that (i) each noise variable* N_i *is a source in the graph (no incoming edges) and is only connected through outgoing edges to corrupted attributes and (ii)* S *and all* A_i^* *are sinks.*

Note that the binary variable S determines whether a tuple will be included in the corrupted dataset or not. This can be used to model selection bias. For example, consider a medical dataset where patients with gender attribute G equal to female have a certain chance (modelled as a noise variable $N_{selbias}$) to be excluded, because of a computer error in the gynecological ward. This corresponds to a graph fragment $N_{selbias} \rightarrow S \leftarrow G$. Given a dependency graph G, which specifies the relationships between the original attributes and how the noise variables and the original attributes affect the corrupted attributes, we now formalize how each corrupted attribute $A^* \in \mathbf{A}^*$ is computed. Rather than defining a single function, we

introduce the concept of a *pattern corruption template*, a parametric family of functions that can model diverse error patterns. Each template is associated with a *selection pattern*, a conjunction of range conditions over the values of the parents of a corrupted attribute. For a given tuple and attribute, the value of the attribute will be corrupted if the pattern evaluates to true for this tuple. The rationale for using templates with parameters is that our system that searches for effective corruptions can be used to find parameter settings for a corruption template that most degrade the performance of a model.

Definition 2 (Pattern Corruption Function and Template). *For an attribute $A \in \mathbf{A}$ to be corrupted into A^*, a* pattern corruption function *is a pair (F, ϕ) where:*

$$F : \text{Dom}(\boldsymbol{Pa}(A^*)) \longrightarrow \text{Dom}(A^*),$$

where $\boldsymbol{Pa}(A^*) \subseteq \mathcal{V}$ is the parent set of A^* as specified by the dependency graph G. Furthermore, ϕ is a conjunction of range conditions on the attributes in $\boldsymbol{Pa}(A^*)$. For a given tuple t:

$$\phi(t) = \bigwedge_{A \in \boldsymbol{Pa}(A^*)} \Big(l_A \leq t(A) \leq u_A\Big),$$

For a clean tuple t, the corruption function (F, ϕ) is used to compute the corrupted value of attribute A^* in the corresponding corrupted tuple t^*:

$$t^*[A^*] = \begin{cases} F\left(t[\boldsymbol{Pa}(A^*)]\right), & \text{if } \phi(t) \\ t[A] & \text{otherwise.} \end{cases}$$

A corruption template *$\mathbb{F}$ for A^* defines a parametric family of corruption functions:*

$$\mathcal{F} : \text{Dom}(\Theta) \longrightarrow \{F, \phi\},$$

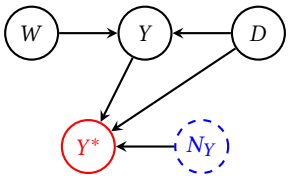
where Θ is a set of parameters controlling the behavior of the corruption process. For all $A \in \boldsymbol{Pa}(A^*)$, the bounds l_A and u_A are parameters in Θ. By specifying settings θ for Θ, the template $\mathcal{F}$ is instantiated into a concrete corruption function $\mathcal{F}(\theta) = (F, \phi)$.

Note that the range conditions used in patterns also allow for equality conditions ($l_A = u_A$) and one sided comparisons. For convenience we will write such conditions as $A = c_A$ and $A \leq c_A$. Note that while corruption functions are deterministic, randomness in the corruption process is modeled through the noise variables whose values are sampled from a probability distribution. Continuing with the selection bias example from above, we model the exclusion of female patients using a pattern $\phi_S : G = female$ and corruption function F_S with parameter p_S which determines the probability of exclusion:

$$F_S(G, N_{selbias}) = \begin{cases} 0 & \text{if } N_{selbias} \leq p_S \\ 1 & \text{otherwise} \end{cases}$$

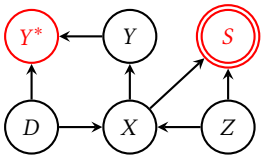
Combining dependency graphs, corruption function templates, and distributions for noise variables we now formally define data corruption process templates (DCPTs) and the concrete data corruption processs (DCPs) that result from applying bindings for the parameters of the template. With the exception of values for noise variables, a DCP fully specifies the transformation of D into $\tilde{D}$ by systematically altering each original attribute based on its specified corruption mechanism and filters tuples based on the value of S.

Definition 3 (Data Corruption Process). *A* data corruption process template *$\boldsymbol{\mathcal{M}} = (G, \mathbb{F}, \Theta)$ is a tuple consisting of:*



D : Demographics
W : Credit, employment, etc
Y : Actual repayment
Y^* : Observed repayment
N_Y : Noise variable

Figure 3: Dependency graphs for missing labels.



D: Demographics
X: Socio-cultural traits
Z: Zipcode
S: Police patrol indicator
Y: Actual crime commitment
Y^*: Observed crime commitment

Figure 4: Dependency graphs for label errors and selection bias. For simplicity, we omit the noise variables.

- *Dependency graph G*
- *Corruption templates $\mathbb{F}$ with a* compatible *template $\mathcal{F}_{A^*}$ for each corrupted attribute $A^* \in \mathbf{A}^*$ and a template $\mathcal{F}_S$ for S, where compatibility requires that $\mathcal{F}_{A^*}$ depends only on $\boldsymbol{Pa}(A^*)$,*
- *A parameter set Θ consisting of Θ_{A^*} for each template, used to instantiate $\mathcal{F}_{A^*}$ into a specific corruption function F_{A^*}. Additionally, Θ contains noise variable distributions $\Omega = \{\omega_1, \ldots, \omega_m\}$, where ω_i is the probability distribution for N_i.*

Given bindings θ for the parameters Θ of a DCPT $\boldsymbol{\mathcal{M}}$, a concrete data corruption process *$\mathcal{M} = \boldsymbol{\mathcal{M}}(\theta)$ is derived from $\boldsymbol{\mathcal{M}}$ by applying the bindings to the corruption templates in $\mathbb{F}$ and associating N_i with ω_i.*

Given a DCP $\mathcal{M}$, the corrupted dataset $\tilde{D}$ is generated as follows. For each tuple, values for each noise variables N_i are first sampled from its distribution ω_i. The selection function F_S is then evaluated using $\mathbf{Pa}(S)$ to determine whether the tuple is included in $\tilde{D}$. If the tuple is included, each corrupted attribute A^* is computed by (i) evaluating ϕ_{A^*} on t and if it evaluates to true, apply F_{A^*} using $\mathbf{Pa}(A^*)$ to compute the corrupted value. This is done in topological order wrt. the dependency graph G, ensuring that each attribute is processed only after its parent attributes have been evaluated. Note that two applications of the same DCP $\mathcal{M}$ may yield different corrupted datasets due to the randomness injected into the process by the noise variables. By varying templates, parameters, and noise distributions, DCPs model a broad range of data-quality issues, including missingness, label errors, and selection bias, as well as complex multi-attribute dependencies.

We illustrate the application of our framework by modeling two key corruption scenarios: missing data and a compound case involving both label errors and selection bias. These examples demonstrate how corruption processes can be systematically defined using structured dependencies, where missing values arise due to observed and latent factors, and label errors interact with selection mechanisms to shape data availability. The same approach extends naturally to other complex corruption patterns, such as outliers, duplication effects, and interactions between multiple error types.

Example 1 (Missing Data in Finance). *The dependency graph in Figure 3 models a missing not at random (MNAR) scenario where*

missingness in the observed repayment outcome Y^* depends on the actual repayment status Y and demographic factors D. Y is corrupted using the following template with parameters $\{p_Y, d, y\}$:

$$F_{Y^*}(Y, D, N_Y) = \begin{cases} \perp & \text{if } N_Y \leq p_Y \\ Y & \text{otherwise} \end{cases} \qquad \phi_{Y^*} : D = d \wedge Y = y$$

where N_Y is a noise variable taking values in $[0, 1]$ with a uniform distribution. The value of N_Y is compared against the parameter p_Y to determine whether Y^* is missing. For instance, setting $p_Y = 0.95$, d = minority, and y = reject means repayment information is missing with 95% probability for minorities with rejected applications and not missing in any other subpopulation.

Example 2 (Compound Errors in Predictive Policing). *Figure 4 shows a dependency graph for predictive policing, where label errors and selection bias coexist. The actual crime Y is influenced by socio-cultural traits X and demographics D, while the observed crime Y^* is subject to label errors that depend on D and Y. Selection bias arises from police patrols S, which determine whether data is collected and are influenced by X and geographic region Z. The label is corrupted using a template with parameters $\{p_Y, d_Y, y_Y\}$:*

$$F_{Y^*}(Y, D) = \begin{cases} 1 - Y & \text{if } N_Y \leq p_y \\ Y & \text{otherwise} \end{cases} \qquad \phi_{Y^*} : D = d_Y \wedge Y = y_Y$$

where N_Y is uniformly sampled from $[0, 1]$. The parameters d_Y and y_Y specify the subpopulation and label values affected.

Selection bias is modelled as a template with parameters $\{p_S, z_S, x_S\}$:

$$F_S(N_S) = \begin{cases} 0 & \text{if } N_S \leq p_S \\ 1 & \text{otherwise} \end{cases} \qquad \phi_S : Z = z_S \wedge X = x_S$$

where $S = 0$ indicates that data was not collected (police did not patrol). The parameter p_S determines the selection probability, while z_S and x_S define the region and subpopulation ignored by police.

Extension to Multi-class Setting. Example 2 describes the label error under a binary class setting. However, Savage also supports the multiclass label error, which is achieved by segmenting the $N_Y \leq p_y$ part into multiple sub-intervals, each representing one class to be changed to.

4 ADVERSARIAL DATA CORRUPTION

In this section, we develop algorithms for identifying worst-case corruption mechanisms that degrade ML pipeline performance within realistic constraints. Given a set of candidate DCPTs $\mathbb{M}_{\text{feasible}}$ provided by the user, our objective is to determine the most adversarial DCP $\mathcal{M}^\dagger$ that minimizes a performance metric while adhering to structural and domain constraints. That is, there exists some DCPT $\boldsymbol{\mathcal{M}} \in \mathbb{M}_{\text{feasible}}$ and bindings θ for the parameters of $\boldsymbol{\mathcal{M}}$ such that $\mathcal{M}^\dagger = \boldsymbol{\mathcal{M}}(\theta)$. Abusing notation we will sometimes write $\mathcal{M} \in \mathbb{M}_{\text{feasible}}$ to denote that there exists $\boldsymbol{\mathcal{M}} \in \mathbb{M}_{\text{feasible}}$ and θ for $\boldsymbol{\mathcal{M}}$ such that $\boldsymbol{\mathcal{M}}(\theta) = \mathcal{M}$. As mentioned previously, the user has full control over the specificity of the candidate DCPs ranging from letting our approach select the parameters for a fixed corruption function template to searching over a wide range of dependency graphs and candidate corruption templates. We define this as an optimization problem over the space of DCPs using bi-level optimization to separate template selection from parameter tuning.

Setting. We are given a training dataset $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \text{Dom}(\mathbf{X})$ represents the input features of a tuple, and $y_i \in \text{Dom}(Y)$ denotes the corresponding label. An ML pipeline $\mathcal{A}$ processes D_{train} through three stages: preprocessing, model training, and post-processing. As we utilize blackbox optimization techniques, our solution supports arbitrary pipelines. The pipeline produces a model h based on D_{train}, which is evaluated on a separate test dataset $D_{\text{test}} = \{(x_i, y_i)\}_{i=1}^M$. The performance of the model is measured using a task-relevant metric $\Psi(h, D_{\text{test}})$, such as accuracy, mean squared error, or fairness measures like demographic parity or equal opportunity. Without loss of generality we assume that higher values of Ψ indicate better performance. To evaluate the effectiveness of a candidate DCP $\mathcal{M}$ in degrading Ψ, we have to rerun $\mathcal{A}$ on $\tilde{D} = \mathcal{M}(D_{\text{train}})$ to get model $h_{\tilde{D}}$ and reevaluate the metric $\Psi(h_{\tilde{D}}, D_{\text{test}})$.

Adversarial Data Corruption. Given $\mathbb{M}_{\text{feasible}}$, our objective is to identify the *most adversarial DCP* $\mathcal{M}^\dagger$ that minimizes the performance metric selected by the user. Note that for a DCP $\mathcal{M}$, $\mathcal{M}(D_{\text{train}})$ is not deterministic as we sample values for the noise variables for each tuple. Thus, we optimize for the DCP with the lowest expected performance:

$$\mathcal{M}^\dagger = \underset{\mathcal{M} \in \mathbb{M}_{\text{feasible}}}{\arg\min}\ \mathbb{E}\left[\Psi\Big(\mathcal{A}(\mathcal{M}(D_{\text{train}})), D_{\text{test}}\Big)\right]. \tag{1}$$

Remark. The term adversarial in $\mathcal{M}^\dagger$ does not imply that the identified corruption DCPs are rare or extreme. Rather, it refers to a DCPs that maximally degrades model performance while being within the bounds of what the user considers realistic. That is, based on the user's background knowledge about possible types of data errors in their domain, we determine the worst case impact on the model performance that can be expected for these types of errors. The user can provide such background knowledge as input in form of a dependency graph and, potentially also corruption function templates. However, our approach does not require these inputs to be provided, but can also search for dependency graphs and select templates autonomously. By using interpretable patterns, the user can easily judge whether a DCP is realistic and if necessary rerun the system excluding patterns they deem to be unrealistic. In contrast to the simple error injection techniques used in past work that evaluates the impact of data quality issues on ML tasks, our approach can ensure the user that their pipeline is robust against realistic worst-case errors.

4.1 Bi-level Optimization Formulation

Exploring all candidate DCPs – selecting a dependency graph and corruption template, and bindings for their parameters – is infeasible. To manage this complexity, we assume a predefined error type (missing values (MVs), label errors, or selection bias) and restrict the search to DCPTs for this error type that corrupt a single *target attribute*: A^* for MVs, Y^* for label errors, or S for selection bias. Note that the error type determines the corruption function template except for the pattern. Furthermore, we assume that each corrupted attribute A_i^* is associated with a single noise variable N_i. As discussed in Section 3, a DCPT defines a structured corruption

mechanism where attributes in the pattern ϕ (used to select the corrupted subpopulation) and used as parameters to the corruption function template correspond to the parents of the target attribute in the dependency graph G, which we assume WLOG includes only these dependencies. Thus, for a given error type, determining $\mathcal{F}$ except for the pattern, we have to choose the subset of original attributes to be used in the pattern. Despite these restrictions, the number of candidate DCPTs remains exponential in the number of attributes as any subset of attributes can be used as the parents of a corrupted attribute, and for each DCPT there may be a large number of possible parameter settings. To address this, we adopt a *bi-level optimization* framework, where the *upper level* selects the optimal DCPT, and the *lower level* tunes parameters, such as the fraction of the selected subpopulation to corrupt.

Structural Components and Corruption Mechanisms. For a dataset D and target attribute A^*, let $\mathbb{M} = \mathbb{F}[D, A^*] \times \mathbf{\Theta}[\mathbb{F}] \times \mathrm{Dom}(\Omega)$ is the space of all corruption mechanisms using DCPs for the input dataset D that above the restrictions mentioned above and their possible parameter settings which include the parameters controlling distributions for all noise variables.

Constraints. Additionally, we allow the specification of further constraints on the candidate space $\mathbb{M}$. Typical constraints include capping the expected fraction of corrupted tuples via $\mathbb{E}[\sum_{i=1}^{N} \mathbf{1}\{t_i \neq t_i^*\} \leq k]$, enforcing valid domain relationships, and bounding corruption parameters to plausible intervals. We use $\mathbb{M}_{\text{feasible}} \subseteq \mathbb{M}$ to denote the resulting pruned search space.

Bi-level Objective. The optimization objective from Equation (1) can be rewritten into bi-level optimization problem:

$$\min_{\mathcal{F} \in \mathbb{F}[D, A^*]} \left\{ \min_{\Theta \in \mathbf{\Theta}[\mathbb{F}]} \Psi\big(\mathcal{A}(\mathcal{M}(D_{\text{train}})), D_{\text{test}}\big) \right\} \text{s.t. } \mathcal{M} \in \mathbb{M}_{\text{feasible}}.$$

At the **upper level**, the pattern ϕ of the DCPT is selected (if the error type and, thus, corruption template is fixed). The **lower level**, determines settings θ for the parameters $\Theta_{\mathcal{F}}$ (e.g., probabilities or thresholds) and noise distributions Ω for a given corruption template $\mathcal{F}$ to minimize the performance metric Ψ. Note that we have dropped the expectation in the bi-level formulation. This is a heuristic choice motivated by the fact that *BO* we use for the lower level has been successfully applied in domains where the objective for a solution may be uncertain. This bi-level formulation balances expressive corruption scenarios with computational feasibility.

4.2 Solving the Bi-level Optimization Problem

The bi-level optimization process alternates between the **upper level**, which selects a pattern and the **lower level**, which tunes parameters for the selected DCPT to maximize the degradation of model performance. By iteratively alternating between these two levels, we efficiently navigate the search space while adhering to the constraints defining $\mathbb{M}_{\text{feasible}}$.

Overall Algorithm. Algorithm 1 provides a high-level overview of the alternating approach. The upper level explores pattern candidates using beam search [49], while the lower level applies Bayesian optimization (BO) to refine parameters for each candidate.

4.2.1 Beam Search for Structural Exploration. To address the exponential size of the search space for DCPTs, we employ *beam search* [49], a heuristic search algorithm that balances exploration and exploitation by retaining and expanding only the most promising candidates at each iteration. We maintain two data structures: (i) a set of candidate corruption function templates $\mathbb{F}_{cand}$ that will be evaluated in the current iteration and a beam $\mathcal{B}$ that contains triples $(\mathcal{F}, \mathcal{M}, \Psi)$ where $\mathcal{F}$ is one of the templates we evaluated in the current or previous iterations, $\mathcal{M}$ is the best DCP we have found in the lower level optimization by tuning parameters of $\mathcal{F}$ and Ψ is the performance of the model trained on $\mathcal{M}(D_{\text{train}})$. We initialize $\mathbb{F}_{cand}$ with the set of all single attribute patterns (recall that the corruption function $F_{\mathcal{F}}$ is determined by the error type and we only optimize over the pattern $\phi_{\mathcal{F}}$ of $\mathcal{F}$). Thus, in our case the beam search is over which attributes to use in the patterns (Recall that here we assume that each corrupted attribute A_i^* is associated with an independent noise variable N_i). At each iteration d, beam search maintains a beam $\mathcal{B}$ of size B. Each candidate $\mathcal{F} \in \mathbb{F}_{cand}$ is evaluated by invoking the lower-level optimization (Section 4.2.2), which tunes parameters including selecting noise distributions $\Theta \in \mathbf{\Theta}[\mathbb{F}]$ to maximize the degradation of the performance metric Ψ. Once all DCPTs in the current candidate set $\mathbb{F}_{cand}$ have been evaluated and added to the beam, we only retain the top-B performers. The beam's templates after pruning are then expanded by extending the patterns of each current DCPTs in all possible ways with a new attribute. These are the candidate templates for the next iteration. As an example, consider the following search. Starting with {Work}, the beam may generate candidates {Work, Age}, {Work, Gender}, and {Work, Race}, each evaluated based on their impact on the performance metric. Beam search terminates when either no further degradation in Ψ is observed (an iteration did not improve the best solution found so far) or a maximum beam depth $d_{\max}$ has been reached. By prioritizing the most promising candidates at each step, beam search provides a computationally efficient approach to identifying high-impact structural corruption mechanisms.

Algorithm 1: Alternating Bi-level Optimization

Input: Training dataset D_{train}, test dataset D_{test}, ML pipeline $\mathcal{A}$, feasible parameter space $\mathbf{\Theta}[\mathbb{F}]$, feasible set $\mathbb{M}_{\text{feasible}}$, beam width B, number of BO iterations τ, max beam depth $d_{\max}$.

```
B = ∅, F_cand ← DETERMINESEEDS(D_train)                    /* Init beam */
for d = 1 to d_max do
    B_old ← B
    foreach F ∈ F_cand do
        M ← BO(F)    /* Algorithm 2 optimizes parameters */
        Ψ ← Ψ(A(M(D_train)), D_test)
        B ← B ∪ {(F, M, Ψ)}
    B ← TOP-K(B, B)
    if ¬ IMPROVES(B_old, B) then
        BREAK                          /* Terminate if no improvement */
    F_cand ← EXPAND(B)                              /* Expand patterns */
return arg min_{(F,M,Ψ)∈B} Ψ                  /* Return optimal M from B */
```

4.2.2 Bayesian Optimization for Parameter Tuning. For a fixed DCP $\mathcal{F}$, optimizing parameters $\theta \in \mathbf{\Theta}[\mathcal{F}]$ is typically a non-convex optimization problem. Furthermore, it requires evaluating the performance of a parameter setting by running the black-box ML pipeline. Bayesian optimization (BO) [51] is well-suited for this setting, as

it balances exploration and exploitation to efficiently locate high-impact parameter configurations and can be applied in scenarios when the quality of a solution is uncertain to some degree.

We employ the *Tree-Structured Parzen Estimator* (TPE) [4], a BO algorithm that models the parameter space using density estimators. During initialization, TPE randomly samples a few sets of parameters and estimates their corresponding performance metric Ψ. At each iteration, TPE separates the set of parameters based on their Ψ values, and fits a probability density function (PDF) for *promising parameters* that result in low Ψ, denoted as $g(\theta)$, and *poor parameters* with high Ψ, denoted as $l(\theta)$. Then for the next set of parameters to evaluate, TPE chooses the one that maximizes the likelihood ratio: $\theta_t = \arg\max_{\theta \in \mathcal{P}(\mathcal{F})} \frac{g(\theta)}{l(\theta)} = \arg\max_{\theta \in \boldsymbol{\Theta}[\mathcal{F}]} \frac{\Pr(\theta|\Psi \leq \Psi^*)}{\Pr(\theta|\Psi > \Psi^*)}$, where Ψ^* is a quantile threshold of past performance. Algorithm 2 shows the full procedure.

Algorithm 2: TPE-based Parameter Tuning for Adversarial Mechanisms

Input: Training data D_{train}, test data D_{test}, pipeline $\mathcal{A}$, DCPT $\mathcal{F}$, feasible parameter space $\boldsymbol{\Theta}[\mathcal{F}]$ and $\text{Dom}(\Omega)$, feasible set $\mathbb{M}_{\text{feasible}}$, # iterations τ.

Initialize TPE densities g, l for parameter space $\boldsymbol{\Theta}[\mathcal{F}]$)
for $t = 1$ **to** τ **do**
  $\theta_t \leftarrow \arg\max_\theta \frac{g(\theta)}{l(\theta)}$ /* Sample next parameters */
  $\mathcal{M} = \mathcal{F}(\theta_t)$ /* Apply parameters */
  Project $\mathcal{M}$ to $\mathbb{M}_{\text{feasible}}$
  $\Psi_t = \Psi(\mathcal{A}(\mathcal{M}(D_{\text{train}})), D_{\text{test}})$ /* Evaluate parameters */
  Update TPE densities g, l based on Ψ_t
return $\arg\min_t \Psi_t$

4.2.3 Implementation and Efficiency Enhancements. Although the bi-level approach finds adversarial corruption mechanisms effectively, we integrate the following strategies to further enhance efficiency and scalability:

Heuristics. During beam search, we impose domain-informed heuristics to avoid unproductive expansions. For instance, pattern-based templates must always include the target attribute(s) (e.g., in Missing-Not-At-Random settings) and the label attribute, which causes a compound of covariate shift and concept drift. We also limit the number of attributes used in patterns to prevent overly sparse subpopulations and enforce feasibility rules that reflect domain constraints (e.g., compatible attribute interactions). By restricting the structural search in this manner, we prune large portions of the search space while preserving high-impact corruption mechanisms.

Knowledge Reuse and Warm-Starting. To reduce computational overhead, we reuse structural and parametric insights gleaned from simpler pipelines or smaller data samples. This reuse, or "warm-starting", leverages the observation that many core properties of adversarial corruption mechanisms remain applicable across different dataset scales and pipelines. For instance, dependency graphs and corruption templates identified with a lightweight model can serve as valuable initial structural candidates when transitioning to a more computationally intensive pipeline. Likewise, parameter distributions (e.g., from TPE density estimators) learned on smaller data can provide an effective initialization for BO on larger data, thereby expediting convergence.

Dataset	# rows	# cols	Label	Task
Adult	45K	3	Income>$50K	Classification
Employee	4.7K	9	Resignation	Classification
Credit Card	30K	8	Default	Classification
India Diabetes	905	17	Type 2 Diabetes	Classification
SQF	48K	14	Frisk	Classification
HMDA	3.2M	8	Loan Approval	Classification
Diabetes	442	10	Severity	Regression

Table 2: Datasets and ML tasks.

Algorithm	Targeted Error Types
Imputers [38]	missing values
BoostClean [24]	missing values , selection bias, label errors
Diffprep [27]	missing values, outliers
H2O [7]	missing values
AutoSklearn [12]	missing values, selection bias

Table 3: Data cleaning algorithms.

5 EXPERIMENTS

In the experiments, we answer the following research questions: **Q1:** How do data errors affect the accuracy and fairness of models trained on cleaned datasets prepared with state-of-the-art data cleaning algorithms? Furthermore, are methods sensitive to particular data corruption processes and types of errors (Section 5.2)? **Q2:** Can robust learning algorithms produce robust models over dirty data and which characteristics of the data corruption process affect their success? Do the guarantees of uncertainty quantification (UQ) methods still hold when the data is subject to systematic errors (Section 5.3)? **Q3:** How robust are models when data corruption is adversarial and systematic compared to non-adversarial settings as used in prior experimental studies on the impact of data quality and cleaning on model robustness [23] (Section 5.4)? **Q4:** What is the effectiveness and performance of Savage and its components? And how effective is Savage compared to state-of-the-art data poisoning techniques? (Section 5.5). All our experiments are performed on a machine with an AMD Opteron(tm) 4238 processor, 16 cores, and 125G RAM. Experiments are repeated 5 times with different random seeds, and we report the mean (error bars denote standard deviation).

5.1 Setup

Datasets and Data Errors. As shown in Table 2, we conduct experiments primarily on six representative ML tasks: classification (*Adult, Employee, Credit Card, India Diabetes*, and *SQF* datasets) used for the evaluation of data cleaning (data cleaning) and robust learning algorithms, and one regression dataset (*Diabetes*) used for evaluating UQ methods. We use the *India Diabetes* dataset for a case study on error patterns as it has been analyzed in related work [23]. Although Savage supports corruption with a wide range of errors, in the paper, we focus on three common data errors: MVs, selection bias (and sampling error), and label errors. Unless explicitly mentioned, the target column for injecting MVs is automatically selected during beam search and we attack only the training data.

Algorithm	Objective	Targeted Error Types
Reweighing [22]	Debiasing	✗
LFR [53]	Debiasing	✗
Fair Sampler [41]	Debiasing	label errors
Fair Shift [42]	Debiasing	correlation shift
Split CP [26]	UQ	✗
Split-MDA CP [52]	UQ	missing values

Table 4: Debiasing and UQ algorithms.

Algorithms and Models. Table 3 lists the data cleaning solutions used in our experiments. For each approach, we specify which types of errors are targeted by the method. To cover a wide range of methods, we included automated systems like BoostClean [24], H2O [25], Diffprep [27] , and AutoSklearn [12]. We also use popular implementations of standard imputation techniques, including imputing with the mean and median value of a feature, and advanced methods such as KNN imputation and iterative imputation [38].

We also evaluate the robustness of techniques that aim to reduce biases of models or quantify the uncertainty in model predictions. To evaluate how well debiasing and uncertainty quantification (UQ) algorithms handle data errors, we contrast approaches that were explicitly designed to handle data errors with those that do not. We evaluate the techniques listed in Table 4. The purpose of debiasing techniques is to reduce biases in predictions made by a model. Specifically, for the debiasing tasks, we test two widely used preprocessing methods: Reweighing [22] and LFR [53], as well as Fair Sampler [41], which addresses noisy labels, and Fair Shift [42], which handles correlation shifts where the correlation between the label and sensitive attribute changes. For UQ, we use two conformal prediction (CP) techniques: Split CP [26] and CP-MDA-Nested [52]. For regression, CP takes as input a significance level α in $[0, 1]$ and returns, for each data point, a prediction interval such that, with $1 - \alpha$ probability, the data point's true label is within the interval.

We evaluate how these cleaning, debiasing, and UQ algorithms are impacted by systematic data corruption process (DCP) by training ML models on data prepared using these methods. We consider the following types of models: *logistic regression*, *decision trees*, *random forest*, and *neural network*, which is a feed-forward neural network with 1 hidden layer containing 10 neurons. The parameters of models are specified in the code repository [2].

As discussed in section 2, Savage and indiscriminate data poisoning both attack a model's performance by corrupting the training data. Even though the motivations (and requirements) of these two lines of work are different, to evaluate the raw effectiveness in degrading model performance, we compare Savage against the state-of-the-art Gradient Cancelling [32] (GradCancel) and Back Gradient [35] (BackGrad) poisoning attacks.

Metrics. We measure the amount of errors in a dataset as the percentage %E of the rows that are affected by at least one errors. For instance, for missing values, 50% would indicate that 50% of the rows contain one or more missing values. To measure model performance, we use area under the curve (AUC) and F1 score (F1) for classification tasks and mean-squared error (MSE) for regression tasks. We also measure the bias of a model using standard fairness metrics: statistical parity difference (SPD) [9] and equality of opportunity (EO) [18]. For UQ tasks, we calculate coverage rate.

Dependency Graph Transfer. To deal with the high runtime for automated data-cleaning frameworks such as AutoSklearn and BoostClean, we warm start the search for BoostClean, AutoSklearn, Diffprep, and H2O by transferring the worst-case dependency from the iterative-imputer and then finetune the corruption parameters using TPE.

5.2 Sensitivity of Data Cleaning Methods

We use Savage to inject errors into datasets to attack automated data cleaning techniques (Section 5.2.1), robust fairness algorithms (Section 5.3), and UQ (Section 5.3.2). We vary error percentage (%E) and measure model accuracy and fairness.

5.2.1 Data Cleaning Techniques. The results for data cleaning techniques are shown in Figures 5 to 7, where the red dotted line represents the maximum AUC achieved across all the methods when no errors are present in the data. We use this as a baseline as some of the techniques apply transformations that are beneficial even if no data errors are present.

Varying Data Quality Issues. We first focus on logistic regression, varying the type of data quality issues injected by Savage using the Adult, Credit Card, SQF datasets. For MVs (Figure 5), fewer than 10% missing values in a single column can reduce AUC by over 0.05 for most methods, with the exception of Diffprep on Credit Card. At 30%, the reduction exceeds 0.15 in most cases, except for Diffprep and KNN-imputer on Credit Card, and all methods on SQF, which contains fewer predictive features. Learning-based methods like KNN-imputer and Diffprep are generally more robust than simpler techniques such as mean-imputer. Compared to MVs, selection bias and label errors are more detrimental under the same budget. For this experiment, we exclude imputation methods. On Employee, a 25% corruption budget reduces all methods below 0.45 AUC, comparable to the impact of 50% MVs. Similarly, on Adult, 15% corruption leads to AUC below 0.6. These errors are more harmful as they induce greater shifts in Pr(Label | Covariate), disrupting feature-label dependencies more severely.

> *Key Takeaway:* Data-cleaning techniques are sensitive to small amounts of systematic data corruption. Adaptive techniques like Diffprep are more effective, but also less stable.

Varying Downstream Models. Next, we vary what type of model that is trained using the Adult and Employee datasets (Figure 7). In general, decision trees and neural network exhibit greater variance in performance. For example, on Adult with 30% corruption, the AUC range spans 0.2 for decision trees and 0.3 for neural network, compared to only 0.1 for random forest. This indicates greater sensitivity and potential overfitting in decision trees and neural network. By contrast, random forest demonstrates higher robustness, likely due to its ensemble structure, with consistently higher AUC across corruption levels—particularly on Employee.

> *Key Takeaway:* Decision trees and neural network are more susceptible to data corruption than random forest.

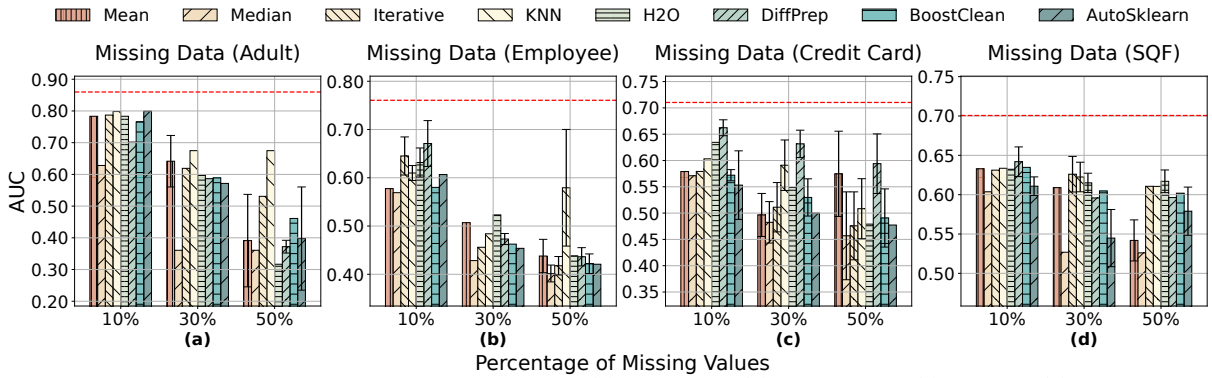


Figure 5: AUC of logistic regression when corrupting Adult (a), Employee (b), Credit Card (c) and SQF (d) datasets with MVs.

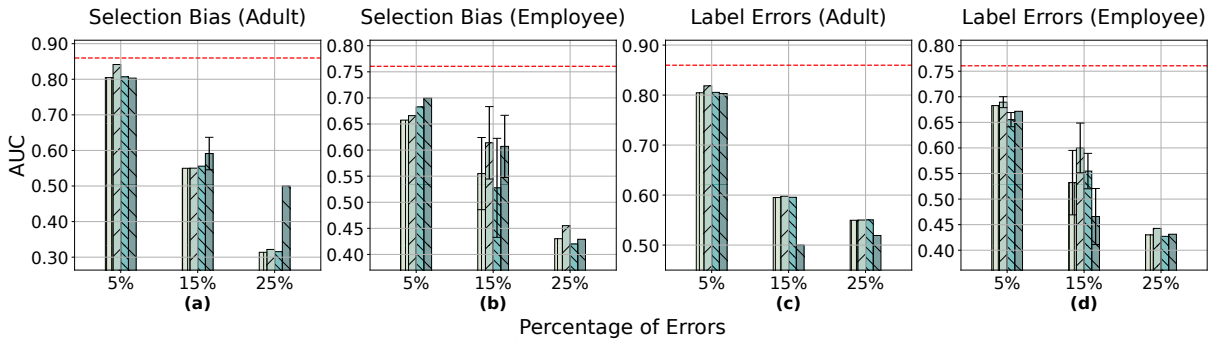


Figure 6: AUC of logistic regression, corrupting Adult (a, c) and Employee (b, d) with selection bias (left) and label errors (right).

5.3 Debiasing and Uncertainty Quantification

We analyze the robustness of fair ML and UQ under systematic label errors and selection bias.

5.3.1 Debiasing. We evaluate debiasing methods on the Adult dataset using logistic regression. Specifically, we test Fair Sampler [41], which addresses label errors, and Fair Shift [42], designed for correlation shifts. To simulate their application scenarios, we use SAVAGE to (1) generate systematic label flips for evaluating Fair Sampler and (2) introduce selection bias for assessing Fair Shift. For comparison, we also include Reweighing and LFR, which are not designed for systematic corruption. We use a 10% corruption budget using F1 and unfairness measured by EO. We also report performance of a baseline logistic regression trained on the corrupted data (denoted *Orig.*). Without corruption, all methods achieve F1 $\geq$ 0.471, with regular logistic regression reaching 0.555.

Table 5 shows the results under label errors. Training directly on corrupted data yields a low F1 of 0.32. All methods perform poorly under label errors, with F1 scores below 0.21. Reweighing fails to mitigate bias while LFR produces a poor classifier (F1 = 0.01).

Results for selection bias are shown in Table 6. Similar to label errors, selection bias degrades both fairness and accuracy, though the severity of the impact varies. Fair Shift shows the best balance between accuracy and fairness, achieving F1 = 0.35 and EO = 0.13. Reweighing increases bias (EO = 0.36) at marginally better accuracy (F1 = 0.33). LFR again fails to produce a usable model (F1 = 0.01).

Metrics	Orig.	Reweighing	LFR	FairSampler [41]
F1	0.32 ± 0	0.21 ± 0	0.01 ± 0.02	0.2 ± 0.01
EO	0.22 ± 0	0.49 ± 0.01	0.01 ± 0.01	0.06 ± 0

Table 5: Robustness of debiasing methods under label errors targeting fairness measured by EO (budget: 10%).

Metrics	Orig.	Reweighing	LFR	FairShift [42]
F1	0.38 ± 0	0.33 ± 0	0.01 ± 0.02	0.35 ± 0
EO	0.22 ± 0	0.36 ± 0.01	0.01 ± 0.01	0.13 ± 0.02

Table 6: Robustness of debiasing methods under selection bias targeting fairness measured by EO (budget: 10%).

Key Takeaway: While debiasing methods like Fair Sampler and Fair Shift can reduce unfairness under systematic corruptions, they often fail to preserve classification performance.

5.3.2 Uncertainty Quantification. We also analyze the robustness of conformal prediction (CP) when the training data contains MVs. In CP the user provides a target coverage $1-\alpha$ and the CP approach computes a prediction interval for a test data point such that the ground-truth label is guaranteed to be within the interval with probability at least $1-\alpha$. We use the Diabetes dataset and employ SAVAGE using a budget of 30% to generate MVs that break the coverage guarantee of CP. We benchmark Split CP, the standard split conformal prediction, and CP-MDA-Nested [52], an extension designed for robustness to MVs. Tables 7 and 8 show the results for different target coverages $1-\alpha$. Both methods achieve the target coverage when no data errors are present. However, with less than

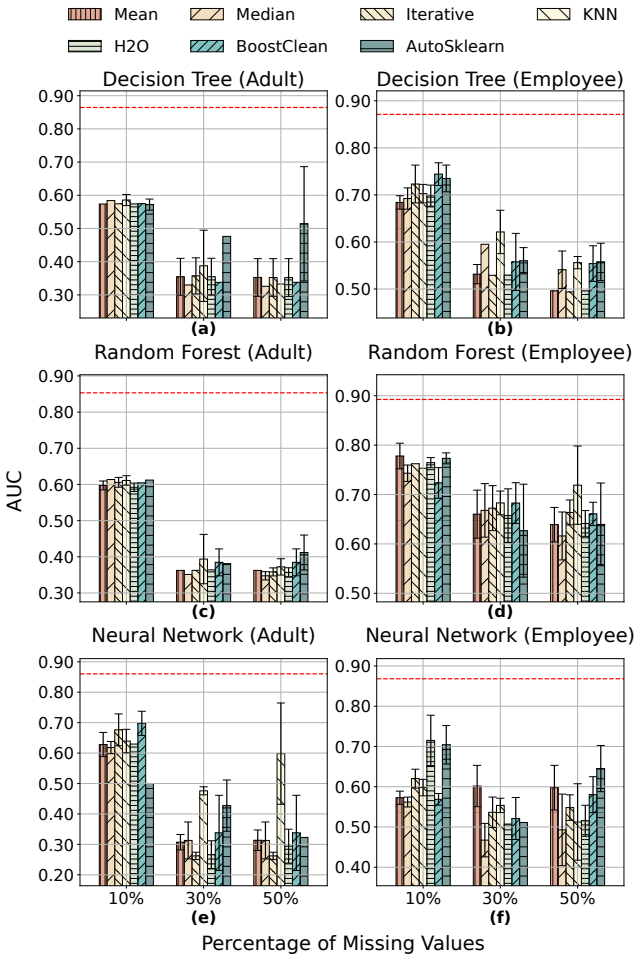


Figure 7: AUC of decision trees (first row), random forest (second row), and neural network (third row) when injecting MVs into the Adult (left) and Employee (right).

Algorithm	Coverage (%)	Missing Rate (%)
Split CP (clean)	96.2 ± 1.1	0
CP-MDA-Nested (clean)	95 ± 2.3	0
Split CP (corrupted)	91.7 ± 2.1	22.7 ± 6.1
CP-MDA-Nested (corrupted)	89 ± 2.2	19.6 ± 7.9

Table 7: Actual coverage of Split CP and CP-MDA-Nested with missing values in the training data ($\alpha = 0.05$).

Algorithm	Coverage (%)	Missing Rate (%)
Split CP (clean)	80 ± 5.2	0
CP-MDA-Nested (clean)	81.6 ± 4.8	0
Split CP (corrupted)	70.8 ± 4.4	23.7 ± 6.4
CP-MDA-Nested (corrupted)	73.5 ± 4.2	20.3 ± 7.3

Table 8: Actual coverage of Split CP and CP-MDA-Nested with missing values in the training data ($\alpha = 0.2$).

30% MVs in a single column, the average coverage of both methods dropped by more than 4.5% when the target coverage was 0.95 ($\alpha = 0.05$), and by over 8% for a target coverage of 0.8 ($\alpha = 0.2$). Even though CP-MDA-Nested is designed to handle MVs, it fails to achieve the desired coverage when *systematic* data errors are introduced using Savage. This is primarily due to its reliance on the assumptions that the missingness mechanism is conditionally independent of the label variable which is often violated in real-world scenarios and by the MVs injected by Savage [14].

> *Key Takeaway:* Even CP approaches designed to be robust against MVs fail to maintain the coverage guarantee when MVs are systematic.

5.4 Reevaluating Robustness Claims

Existing benchmarks have investigated the impact of systematic errors using manually specified patterns [23, 46]. To evaluate whether Savage can identify vulnerabilities of ML pipelines overlooked in prior work, we use a setting from *shades-of-null* [23] as an example, comparing their manually specified error patterns from [23] against patterns generated by Savage. We use the India Diabetes dataset, set the error budget to 10%, and use F1 as the target. Note that shades-of-null assumes that both training and testing datasets contain MVss. We conduct a search for adversarial patterns where training data errors are MNAR, while test data errors are missing completely at random (MCAR), which is closest to our previous setting of assuming no errors in the test data. The pattern used by Savage to identify the subpopulation to inject MVs into is:

(Num_Pregnancies ≤ 2) ∧ (Family_Diabetes = No) ∧ (Type_II_Diabetes = yes)

Injecting MVs according to this pattern results in a 0.81 F1. The F1 on the clean dataset is 0.95. However, with the same setting, the pattern tested in shades-of-null reports an F1 of 0.88. This gap is larger when testing with higher error budgets. For instance, with an error budget of 30%, the pattern discovered by Savage leads to an F1 of 0.36, while shades-of-null reported an F1 of 0.87 under this setting. The key difference between the patterns presented in shades-of-null and the one generated by Savage is that Savage also explores patterns that use the label (Type_II_Diabetes, in this example). Although highly adverse, this is a realistic setting that needs to be tested. shades-of-null uses manually created error patterns taking into account signals such as feature correlations and importance. Although these patterns encompass ML practitioners' insights, they do not fully reflect hard real-world cases, e.g., cases where the missingness of values depends on the label.

> *Key Takeaway:* Prior work is overly optimistic, overlooking realistic, but highly adverse, corruption types.

5.5 Efficiency and Effectiveness of Savage

We evaluate the effectiveness of Savage through ablation studies (Section 5.5.1) and comparison with state-of-the-art data poisoning techniques (Section 5.5.2). We also conduct a scalability analysis and discuss the solution quality on large-scale data(Section 5.5.3).

5.5.1 Ablation Studies. We evaluate Savage's components, including beam search for DCPTs and the TPE-based optimization for parameters. We also include the ablation study for the heuristics for filtering dependencies in the extended version [55].

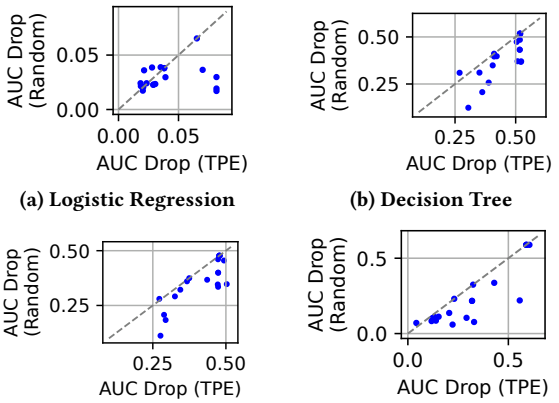
Beam Search. We compare beam search against a baseline that randomly samples and evaluates 100 dependency graphs, selecting the one that causes the greatest reduction in AUC. Both use TPE

Method	Mean Imputer	Median Imputer	Iterative Imputer	KNN Imputer	H2O	Diffprep	BoostClean	AutoSklearn
Random Search	0.8 ± 0	0.79 ± 0	0.8 ± 0	0.8 ± 0	0.61 ± 0.15	0.73 ± 0.11	0.81 ± 0.02	0.75 ± 0.16
Beam Search	0.39 ± 0.14	0.36 ± 0	0.53 ± 0	0.67 ± 0	0.32 ± 0	0.37 ± 0.02	0.46 ± 0	0.4 ± 0.16

Table 9: AUC of logistic regression trained on worst-case data corruptions generated by random search and beam search.

AUC drop	[0, 0.08)	[0.08, 0.16)	[0.16, 0.24)	≥ 0.24	**Total**
Cnt. (no rules)	4885	514	204	16	5619
Cnt. (rules)	1250	375	128	16	1769
Percentage	0.26	0.73	0.63	1	0.31

Table 10: Heuristic pruning of ineffective patterns.



(a) Logistic Regression (b) Decision Tree

(c) Random Forest (d) Neural Network

Figure 8: Decrease in AUC of models trained with corrupted data, where corruption parameters are optimized by TPE (x-axis) and random search (y-axis). Each point represents a random dependency graph.

for the corruption parameter search. This experiment is conducted on the Adult dataset using logistic regression as the downstream model, with 50% budget. The results shown in Table 9 demonstrate that Savage consistently identifies error patterns that result in significantly lower AUC compared to random search. In most cases, the margin of difference is above 0.25 in AUC. The huge gap arises because random sampling operates within a vast search space.

TPE. We demonstrate the effectiveness of the TPE component, which is responsible for finding adversarial corruption parameters given a dependency graph, by comparing it with the random corruption parameter search. To do this, we randomly sample 20 dependency graphs and conduct the parameter search for each. Figure 8 presents the AUC drop of the models trained on the corrupted data discovered by TPE and random search, where each point represents a sampled dependency graph. Savage consistently identifies corruption parameters that lead to higher AUC drop, compared with random search, especially for the most harmful cases that Savage targets during beam search.

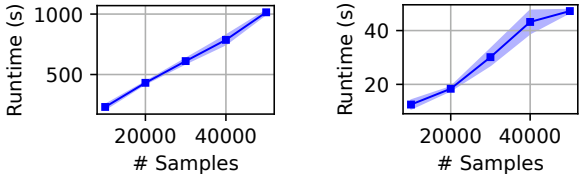
5.5.2 Comparison with Data Poisoning. In the following, we compare Savage with state-of-the-art data poisoning methods, as well as a random baseline (Rand), which conducts random corruption within the budget 100 times and returns the worst case.

Indiscriminate Attack. Since the baselines GradCancel and BackGrad rely on editing features of data, for a fair comparison, we stick to the missing data setting, without introducing selection bias or label errors, which are more detrimental. We leverage

% corruption	10	30	50
Rand	0.85 ± 0	0.85 ± 0	0.85 ± 0
BackGrad	0.83 ± 0	0.67 ± 0	0.46 ± 0.06
GradCancel	0.79 ± 0.04	0.7 ± 0.02	0.45 ± 0
Savage	0.58 ± 0	0.46 ± 0	0.42 ± 0.02

% corruption	10	30	50
Rand	0.86 ± 0	0.86 ± 0	0.86 ± 0
BackGrad	0.84 ± 0.01	0.78 ± 0.01	0.64 ± 0.02
GradCancel	0.84 ± 0.01	0.81 ± 0.01	0.8 ± 0.01
Savage	0.7 ± 0.04	0.34 ± 0.12	0.34 ± 0.12

Table 11: Effect on AUC when applying Savage and indiscriminate data poisoning on logistic regression (top) and neural network (bottom).



(a) Dependency Search (b) Finetuning

Figure 9: Runtime of Savage with varying dataset sizes.

BoostClean for addressing missing data. Table 11 presents the comparison between Savage, BackGrad, GradCancel, and Rand on the Adult data, where Savage consistently discovers more effective corruption than Rand, BackGrad and GradCancel. This is primarily because existing poisoning attacks are typically designed for unstructured data without demographic attributes. As a result, they often overlook structured patterns and rely on random sampling to select points for modification. In contrast, Savage explicitly models systematic, non-random errors and captures their impact, particularly when corruption targets specific subpopulations. In addition, the ineffectiveness of Rand indicates the difficulty of stochastically discovering adverse cases with completely random corruption.

Moreover, poisoning attacks are generally less effective on neural network than on simpler, convex models like logistic regression, failing to reflect the greater sensitivity of neural network to biases and data quality issues. In contrast, the structured corruptions uncovered by Savage better expose this vulnerability.

> *Key Takeaway:* State-of-the-art data poisoning methods often overlook the impact of systematic subpopulation errors, thus showing worse effectiveness than Savage.

5.5.3 Scalability Analysis. We analyze the runtime breakdown of Savage, and discuss the efficiency and effectiveness of Savage when handling large-scale data with the sampling technique.

Runtime Breakdown. In Table 12, we show a breakdown for Savage's runtime evaluated on a 30K sample of the HMDA dataset with logistic regression. The runtime is broken down into two stages, where for automated cleaning techniques, including H2O,

Stage	Mean	Median	Iterative	KNN	H2O	Diffprep	BoostClean	AutoSklearn
Dependency Search	375.4 ± 2.5	370.2 ± 7.7	611.1 ± 19.6	651.1 ± 6.4	611.1 ± 19.6	611.1 ± 19.6	611.1 ± 19.6	611.1 ± 19.6
Finetuning	26.1 ± 0.9	25.8 ± 1	30.1 ± 3	34.8 ± 0.3	12.6 ± 0.4	851.6 ± 1	153.7 ± 2.5	363.4 ± 17

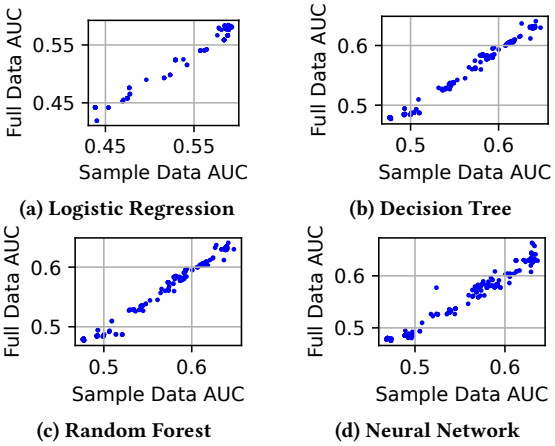
Table 12: Breakdown of Savage's runtime on a 30K sample of HMDA (seconds).

Diffprep, BoostClean, and AutoSklearn, the dependency search involves searching a dependency graph for a proxy method instead of directly running Algorithm 1. The rationale for this approach is that dependency graphs typically translate well between similar cleaning techniques, and this allows us to significantly reduce the search cost by replacing an expensive cleaning technique with a cheaper proxy during search. The finetuning stage involves tuning the corruption parameters using Algorithm 2, which is generally faster than the dependency graph search. However, as demonstrated in the comparison with shades-of-null, selecting the right dependency graph is critical for generating adversarial errors.

Overall, the runtime of Savage is within 20 minutes, even for expensive frameworks such as Diffprep. This benefits from the utilization of proxy models during dependency search. Without this optimization, the search time for Diffprep increases to over 3 hours. We validated the effectiveness of the proxy models by testing dependency search on BoostClean and H2O. Specifically, performing dependency search directly on these frameworks yielded AUC values that differed by less than 0.01 from those obtained using patterns transferred from iterative-imputer. Another important hyperparameter, error budget, is essential for the data corruption process, but does not affect Savage's runtime much, as running the pipeline dominates the runtime of Savage. As a result, Savage's scalability wrt. increased dataset size primarily depends on the scalability of the evaluated pipeline. For instance, most methods, such as the iterative-imputer, have linear time complexity in terms of dataset size, leading to a linear growth of Savage's runtime. For instance, Figure 9 shows the runtime for logistic regression and iterative-imputer, and demonstrates Savage's linear complexity. Other than these factors, the runtime of Savage grows linearly with the beam size and number of BO iterations, as the number of framework evaluations is linear in the number of iterations.

> *Key Takeaway:* The runtime of the evaluated frameworks dominates the runtime of Savage. By utilizing cheaper proxy models for dependency search, Savage achieves an effective and efficient search for adverse corruption mechanisms for time-consuming frameworks.

Handling Large Datasets. For large-scale datasets, the runtime of Savage is typically dominated by the cost of running the ML pipeline itself. According to Figure 9, running Savage on the full HMDA data is expected to take 17.8 hours. To mitigate this, we perform the dependency search phase on a small sample of the data (1%). We then evaluate whether the DCP discovered on the sample is also effective on the full dataset. To this end, we collect all DCPs found during the search phase of iterative-imputer on the sampled HMDA and evaluate the effectiveness of them on the full dataset. As shown in Figure 10, the model's performance on the 1% sample closely matches that on the full dataset across all models. This indicates that the corruption patterns identified on the sample are also harmful at scale. The end-to-end runtime of Savage on the sampled data is under 20 minutes, yet it discovers error patterns that cause over 0.15 AUC drop on the full dataset.



(a) Logistic Regression (b) Decision Tree

(c) Random Forest (d) Neural Network

Figure 10: AUC of models trained with different data corruption mechanisms on sample (x-axis) and full data (y-axis).

> *Key Takeaway:* With sampling, Savage efficiently and effectively identifies adverse data corruptions for the large-scale HMDA data that has over 3.2 million tuples.

5.6 Summary

Our evaluation shows that all studied data cleaning, debiasing, and UQ techniques are highly sensitive to systematic errors, with even small corruptions severely degrading model performance. Certifying robustness requires a system like Savage to generate adversarial errors, as demonstrated by replicating related work. Moreover, debiasing and UQ methods often rely on assumptions that break under systematic corruption, leading to violated guarantees, even for techniques explicitly designed to handle data errors, such as CP-MDA-Nested and Fair Sampler. Savage achieves superior effectiveness and interpretability compared to state-of-the-art indiscriminate data poisoning techniques.

6 CONCLUSIONS

Data quality issues such as missing values and selection bias significantly impact ML pipelines, yet existing evaluation methods often rely on random or manually designed corruptions that fail to capture real-world systematic errors. This work introduces a formal framework for modeling the data corruption process and Savage, a system that automatically generates adversarial corruption mechanisms through bi-level optimization. Savage systematically identifies worst-case corruptions that degrade model performance while adhering to realistic constraints, providing a principled approach for evaluating the robustness of data cleaning, fairness-aware learning, and uncertainty quantification techniques. Our experiments reveal vulnerabilities in existing ML pipelines, demonstrating that current robustness measures are often insufficient against structured corruption.

REFERENCES

[1] 2024. Regulation (EU) 2024/1689 of the European Parliament and of the Council on Artificial Intelligence. https://artificialintelligenceact.eu/article/15/. Article 15: Accuracy, Robustness and Cyber-security.

[2] 2025. Codebase for SAVAGE. https://github.com/lodino/savage

[3] Mohamed Abdelaal, Christian Hammacher, and Harald Schoening. 2023. Rein: A comprehensive benchmark framework for data cleaning methods in ml pipelines. *arXiv preprint arXiv:2302.04702* (2023).

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).

[6] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. 2023. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *Comput. Surveys* 55, 13s (2023), 1–39.

[7] Darren Cook. 2016. *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI.* O'Reilly Media, Inc.

[8] Jimmy Z Di, Jack Douglas, Jayadev Acharya, Gautam Kamath, and Ayush Sekhari. 2022. Hidden poison: Machine unlearning enables camouflaged poisoning attacks. In *NeurIPS ML Safety Workshop*.

[9] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. 2012. Fairness through awareness. In *ITCS*. ACM, 214–226.

[10] Adrien Ehrhardt, Christophe Biernacki, Vincent Vandewalle, Philippe Heinrich, and Sébastien Beben. 2021. Reject inference methods in credit scoring. *Journal of Applied Statistics* 48, 13-15 (2021), 2734–2754.

[11] Chen Xinyun et al. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[12] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *The Journal of Machine Learning Research* 23, 1 (2022), 11936–11996.

[13] Milena A Gianfrancesco, Suzanne Tamang, Jinoos Yazdany, and Gabriela Schmajuk. 2018. Potential biases in machine learning algorithms using electronic health record data. *JAMA internal medicine* 178, 11 (2018), 1544–1547.

[14] John Graham. 2012. *Missing data: Analysis and design. New York, NY: Springer.* https://doi.org/10.1007/978-1-4614-4018-5

[15] Gareth J Griffith, Tim T Morris, Matthew J Tudball, Annie Herbert, Giulia Mancano, Lindsey Pike, Gemma C Sharp, Jonathan Sterne, Tom M Palmer, George Davey Smith, et al. 2020. Collider bias undermines our understanding of COVID-19 disease risk and severity. *Nature communications* 11, 1 (2020), 1–12.

[16] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2022. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. *ICDE* (2022).

[17] Luke Haliburton, Sinksar Ghebremedhin, Robin Welsch, Albrecht Schmidt, and Sven Mayer. 2023. Investigating Labeler Bias in Face Annotation for Machine Learning. *arXiv preprint arXiv:2301.09902* (2023).

[18] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *NIPS*. 3315–3323.

[19] Maliha Tashfia Islam, Anna Fariha, Alexandra Meliou, and Babak Salimi. 2022. Through the data management lens: Experimental analysis and evaluation of fair classification. In *Proceedings of the 2022 International Conference on Management of Data*. 232–246.

[20] Matthew Jagielski, Giorgio Severi, Niklas Pousette Harger, and Alina Oprea. 2021. Subpopulation data poisoning attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3104–3122.

[21] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and information systems* 33, 1 (2012), 1–33.

[22] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems* 33, 1 (2012), 1–33.

[23] Falaah Arif Khan, Denys Herasymuk, Nazar Protsiv, and Julia Stoyanovich. 2024. Still More Shades of Null: A Benchmark for Responsible Missing Value Imputation. *arXiv preprint arXiv:2409.07510* (2024).

[24] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. 2017. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299* (2017).

[25] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML San Diego, CA, USA.

[26] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. 2018. Distribution-free predictive inference for regression. *J. Amer. Statist. Assoc.* 113, 523 (2018), 1094–1111.

[27] Peng Li, Zhiyi Chen, Xu Chu, and Kexin Rong. 2023. DiffPrep: Differentiable Data Preprocessing Pipeline Search for Learning over Tabular Data. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[28] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 13–24.

[29] Heng Liu and Gregory Ditzler. 2021. Data poisoning against information-theoretic feature selection. *Information Sciences* 573 (2021), 396–411.

[30] Brandon Lockhart, Jinglin Peng, Weiyuan Wu, Jiannan Wang, and Eugene Wu. 2021. Explaining inference queries with bayesian optimization. *arXiv preprint arXiv:2102.05308* (2021).

[31] Yiwei Lu, Gautam Kamath, and Yaoliang Yu. 2022. Indiscriminate data poisoning attacks on neural networks. *arXiv preprint arXiv:2204.09092* (2022).

[32] Yiwei Lu, Gautam Kamath, and Yaoliang Yu. 2023. Exploring the limits of model-targeted indiscriminate data poisoning attacks. In *International Conference on Machine Learning*. PMLR, 22856–22879.

[33] Qingwei Luo, Sam Egger, Xue Qin Yu, David P Smith, and Dianne L O'Connell. 2017. Validity of using multiple imputation for" unknown" stage at diagnosis in population-based cancer registry data. *PLoS One* 12, 6 (2017), e0180033.

[34] Sedir Mohammed, Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Sina Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch. 2025. The Effects of Data Quality on Machine Learning Performance on Tabular Data. *Inf. Syst.* 132 (2025), 102549. https://doi.org/10.1016/J.IS.2025.102549

[35] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*. 27–38.

[36] National Institute of Standards and Technology. 2024. *Artificial Intelligence Risk Management Framework (1.0) – Generative AI Profile.* Technical Report NIST AI 600-1. U.S. Dept. of Commerce. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

[37] Judea Pearl. 2009. *Causality*. Cambridge university press.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[39] Jennifer K Plichta, Christel N Rushing, Holly C Lewis, Marguerite M Rooney, Dan G Blazer, Samantha M Thomas, E Shelley Hwang, and Rachel A Greenup. 2023. Implications of missing data on reported breast cancer mortality. *Breast Cancer Research and Treatment* 197, 1 (2023), 177–187.

[40] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable data-based explanations for fairness debugging. In *Proceedings of the 2022 international conference on management of data*. 247–261.

[41] Yuji Roh, Kangwook Lee, Steven Whang, and Changho Suh. 2021. Sample selection for fair and robust training. *Advances in Neural Information Processing Systems* 34 (2021), 815–827.

[42] Yuji Roh, Kangwook Lee, Steven Euijong Whang, and Changho Suh. 2023. Improving fair training under correlation shifts. In *International Conference on Machine Learning*. PMLR, 29179–29209.

[43] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1579–1590.

[44] Donald B Rubin. 1978. Multiple imputations in sample surveys-a phenomenological Bayesian approach to nonresponse. In *Proceedings of the survey research methods section of the American Statistical Association*, Vol. 1. American Statistical Association Alexandria, VA, USA, 20–34.

[45] Svetlana Sagadeeva and Matthias Boehm. 2021. Sliceline: Fast, linear-algebra-based slice finding for ml model debugging. In *Proceedings of the 2021 international conference on management of data*. 2290–2299.

[46] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2021. JENGA-A Framework to Study the Impact of Data Errors on the Predictions of Machine Learning Models.. In *EDBT*. 529–534.

[47] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P Dickerson, and Tom Goldstein. 2021. Just how toxic is data poisoning? a unified benchmark for backdoor and data poisoning attacks. In *International Conference on Machine Learning*. PMLR, 9389–9398.

[48] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in neural information processing systems* 31 (2018).

[49] Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. 1994. Improvements in beam search.. In *ICSLP*, Vol. 94. 2143–2146.

[50] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. 2018. Clean-label backdoor attacks. (2018).

[51] Xilu Wang, Yaochu Jin, Sebastian Schmitt, and Markus Olhofer. 2023. Recent advances in Bayesian optimization. *Comput. Surveys* 55, 13s (2023), 1–36.

[52] Margaux Zaffran, Aymeric Dieuleveut, Julie Josse, and Yaniv Romano. 2023. Conformal prediction with missing values. In *International Conference on Machine Learning*. PMLR, 40578–40604.

[53] Richard S. Zemel, Yu Wu, Kevin Swersky, Toniann Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. In *ICML (3) (JMLR Workshop and Conference Proceedings)*, Vol. 28. JMLR.org, 325–333.

[54] Jiongli Zhu and Babak Salimi. 2024. Overcoming Data Biases: Towards Enhanced Accuracy and Reliability in Machine Learning. *IEEE Data Eng. Bull.* 47, 1 (2024), 18–35.

[55] Jiongli Zhu, Geyang Xu, Felipe Lorenzi, Boris Glavic, and Babak Salimi. 2025. *Stress-Testing ML Pipelines with Adersarial Data Corruption (extended version)*. Technical Report. https://github.com/lodino/savage/blob/main/techreport/techreport.pdf