Constraint-Aware Synthetic Tabular Data Generation via Inter-Column Constraint Discovery with LLM Agents

Jianxing Zhao* **Mao Guan†** **Dongyu Liu***
jxrzhao@ucdavis.edu guanmao771@gmail.com dyuliu@ucdavis.edu

arXiv:2608.15109v1 [cs.AI] 15 Aug 2026

Abstract

Generating structurally valid synthetic tabular data remains difficult: outputs with high statistical fidelity and downstream utility can still violate semantically meaningful domain constraints. We study the discovery and enforcement of three complementary inter-column constraint families—equations, linear inequalities, and logical dependencies. Our unified tool-grounded workflow represents all three as machine-executable hypotheses and applies a common interface for full-table validation, deterministic diagnosis, and counterexample-guided revision. A generator-agnostic post-processor coordinates family-specific repairs on outputs from unchanged tabular generators. Across curated behavioral audits and end-to-end evaluations, the complete workflow improves held-out violation detection over one-shot direct prompting, while postprocessing yields zero measured violations for every retained, applicable constraint, improves downstream utility on most datasets, and largely preserves univariate marginals.

1 Introduction

Synthetic tabular data supports model development, system testing, and analysis when real records are scarce or difficult to use (Shi et al., 2025). Yet outputs with strong distributional similarity and downstream utility can remain structurally implausible: individual records may violate domain constraints (Stoian et al., 2024; Umesh et al., 2025; Afonja et al., 2026). An e-commerce generator may, for example, produce an order whose delivery precedes placement, whose subtotal disagrees with its unit price and quantity, or whose delivery method is incompatible with the product type. Such violations can undermine simulation and decision support, distort downstream models, and make synthetic records distinguishable from real ones through simple logical checks (Long et al., 2025; Afonja et al., 2026; Dhooghe et al., 2026).

Inter-column constraints make structural validity explicit and testable. Prior work either enforces supplied rules through generator-integrated objectives, dependency-aware synthesis, or inference-time refinement (Vero et al., 2023; Stoian et al., 2024; Stoian and Giunchiglia, 2025; Dhooghe et al., 2026), or automates acquisition within a restricted constraint family or synthesis pipeline (Umesh et al., 2025; Afonja et al., 2026; Long et al., 2025). These methods do not jointly address two coupled challenges. First, heterogeneous semantic relationships must be represented as executable hypotheses that can be tested and revised against the complete reference table. Second, enforcement on outputs from unchanged tabular generators must coordinate repairs across shared columns, because one correction can invalidate another or unnecessarily distort the generated distribution. Section 2 reviews these lines of work and formalizes our setting.

We address these gaps with an end-to-end framework for tool-grounded discovery and generator-agnostic enforcement. LLM agents use metadata, column profiles, and sampled records to propose equations, linear inequalities, and logical dependencies in machine-executable forms. Family-specific deterministic tools test every proposal against the complete reference table and return diagnostics and counterexamples for iterative revision. An ordered postprocessor then enforces the retained constraints on outputs from unchanged tabular generators, using distribution-aware repairs while protecting relationships restored earlier. The central contribution is this shared executable interface and coordinated postprocessor: LLM outputs are treated as revisable hypotheses and heterogeneous constraints as an interacting system.

*Jianxing Zhao and Dongyu Liu are with the University of California, Davis.
†Mao Guan is an independent researcher.
[1]Code and data at https://github.com/via-cs/dive-tabular.

We evaluate discovery on three curated benchmarks with two LLM backbones and end-to-end constraint-aware generation on seven public datasets across four generator paradigms. On held-out contrastive audit tasks, the complete executable validation-and-revision workflow achieves higher mean violation-detection accuracy, precision, and recall than one-shot direct prompting in all 18 family–backbone–metric comparisons. Postprocessing yields zero measured violations under every retained, applicable validator; mean TSTR utility improves on four datasets and declines by at most 0.042 on the other three, while mean changes in Column Shapes range from -0.009 to $+0.019$.

Our contributions are:

1. A unified, typed machine-executable interface for equational, linear-inequality, and logical-dependency hypotheses, supporting full-table validation, deterministic diagnosis, and counterexample-guided revision.
2. A generator-agnostic postprocessor that coordinates categorical and numerical repairs across shared columns without backbone access or retraining.
3. Empirical validation across three discovery benchmarks, two LLM backbones, seven end-to-end datasets, and four generator paradigms, demonstrating improved held-out violation detection, zero measured violations under retained, applicable constraints, and largely preserved utility and univariate marginal fidelity.

2 Related Work and Problem Setting

2.1 Tabular Synthetic Data Generation

Let $\mathcal{D}_{\text{real}} = \{\mathbf{x}_i\}_{i=1}^{N}$ be a real tabular dataset with d columns. The domain of column j is denoted by $\mathcal{X}_j$ and may be numerical, with $\mathcal{X}_j \subseteq \mathbb{R}$, or categorical, with $\mathcal{X}_j$ a finite discrete set. Each row (record) $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})$ therefore belongs to the heterogeneous product domain $\mathcal{X} = \prod_{j=1}^{d} \mathcal{X}_j$ and may contain both continuous and discrete values. We treat the rows as i.i.d. samples from an underlying mixed-type joint distribution p_{data} on $\mathcal{X}$. Let $\mathcal{M}$ contain the available metadata, such as the dataset description, column names, data types, and column descriptions.

A tabular generator learns a distribution p_θ over $\mathcal{X}$ from $\mathcal{D}_{\text{real}}$ to approximate p_{data} and produces a synthetic dataset

$$\mathcal{D}_{\text{syn}} = \{\mathbf{x}_i^{\text{syn}}\}_{i=1}^{M}, \qquad \mathbf{x}_i^{\text{syn}} \overset{\text{i.i.d.}}{\sim} p_\theta.$$

Methods for learning p_θ span copula-based statistical models (Patki et al., 2016); deep generators including CTGAN and TVAE (Xu et al., 2019), TableGAN (Park et al., 2018), GOGGLE (Liu et al., 2023), TabDDPM (Kotelnikov et al., 2023), and STaSy (Kim et al., 2023); and language-model generators. GReaT and TapTap learn from serialized rows through fine-tuning or pretraining (Borisov et al., 2023; Zhang et al., 2023), whereas AIGT, EPIC, and Curated LLM use prompting or LLM-guided curation (Zhang et al., 2025; Kim et al., 2024; Seedat et al., 2024). Recent surveys provide broader coverage (Shi et al., 2025; Stoian et al., 2025). Across paradigms, evaluation primarily emphasizes marginal and inter-column distributional similarity and downstream utility, but these aggregate criteria cannot ensure that every record obeys domain semantics (Umesh et al., 2025).

2.2 Constraint-Aware Generation

We study constraint-aware generation, in which a synthetic table should both approximate the real-data distribution and satisfy inter-column constraints implied by dataset semantics and metadata. Prior systems incorporate user-specified logical and statistical rules through differentiable objectives and rejection sampling (Vero et al., 2023); enforce supplied linear-arithmetic constraints during training or inference-time refinement (Stoian et al., 2024; Stoian and Giunchiglia, 2025); or reconstruct dependent features from supplied mappings, dependency graphs, and derivation functions (Umesh et al., 2026; Dhooghe et al., 2026).

We define an instantiated inter-column constraint as a row-separable predicate $c : \mathcal{X} \to \{0, 1\}$ over two or more columns, where $c(\mathbf{x}) = 1$ denotes satisfaction. It is evaluated independently on each record and never compares different records. Our record-level scope is narrower than that of prior systems: it excludes uniqueness and cross-table key constraints supported by SDV (DataCebo, 2026), as well as dataset-level statistical constraints considered by CuTS (Vero et al., 2023).

We focus on three complementary, rather than exhaustive, constraint families: equations capture numerical derivation, linear inequalities numerical feasibility, and logical dependencies categorical admissibility. Together they span numerical and categorical columns and deterministic and set-valued semantics while remaining executable and verifiable at the record level. We use a hypothetical e-commerce order table as a running example.

Equational constraints capture deterministic arithmetic relationships among numerical columns. In the running example, the identity $\texttt{subtotal} = \texttt{unit_price} \times \texttt{quantity}$ states that the subtotal is determined exactly by the unit price and quantity. Our executable checkers support well-defined arithmetic identities without requiring linearity, convexity, or differentiability; related dependency-aware generation encodes such identities through supplied derivation functions (Dhooghe et al., 2026).

Linear inequality constraints define feasible regions over numerical columns, expressing relative bounds, orderings, and capacity limits. For the same table, $\texttt{delivery_time} \geq \texttt{order_time}$ requires delivery to occur no earlier than order placement. Unlike an equation, an inequality generally permits multiple valid assignments, while its convex structure enables tractable joint projection (Stoian et al., 2024; Afonja et al., 2026).

Logical dependency (LD) constraints restrict categorical co-occurrences: for each configuration of determinant columns, the dependent column may take only a specified set of admissible values (Umesh et al., 2025). In this example, $\texttt{zipcode} \rightarrow \texttt{state}$ is an FD: each ZIP code determines one state. A partial rule applies when $\texttt{type} = \texttt{digital}$ and $\texttt{status} = \texttt{paid}$, restricting $\texttt{delivery}$ to $\{\texttt{email}, \texttt{download}, \texttt{in-app}\}$ while permitting multiple valid channels. An FD is a special case of an LD in which each determinant configuration maps to exactly one dependent value; We adopt a broader definition of LD that permits multiple admissible values and applies constraints only to selected determinant configurations, thereby capturing partial and set-valued categorical rules (Umesh et al., 2025, 2026; Dhooghe et al., 2026).

These families require distinct validation and repair operators: equations require tolerance-aware checking and arithmetic reconstruction, inequalities require joint feasibility and numerical projection, and LDs require discrete admissibility maps and conditional replacement. Moreover, constraints can share columns, so repairing one can invalidate another or unnecessarily distort the generated distribution; joint enforcement therefore requires a coordinated repair order rather than independent corrections.

2.3 Constraint Discovery and Enforcement

Automated constraint acquisition remains fragmented. Data-driven methods mine functional or logical dependencies (Umesh et al., 2025); LLM-based systems infer graphical or logical structure within prompt-based or latent-diffusion synthesis pipelines (Liu et al., 2025; Long et al., 2025); and other workflows generate executable data-quality rules or propose, filter, and select linear inequalities (Akella et al., 2025; Afonja et al., 2026). These methods address individual constraint families or couple acquisition to a particular synthesis pipeline.

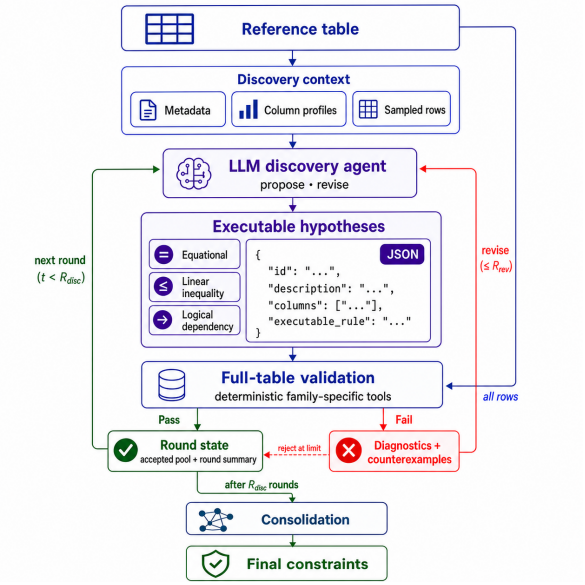

Figure 1: Overview of the LLM-guided constraint discovery workflow.

Given $\mathcal{D}_{\text{real}}$, its metadata $\mathcal{M}$, and an unconstrained synthetic table $\mathcal{D}_{\text{syn}}$, our first objective is to discover executable equational, linear-inequality, and logical-dependency constraints, validate them against all reference rows, and revise failed hypotheses using counterexamples. Our second objective is to postprocess $\mathcal{D}_{\text{syn}}$ so that these constraints are jointly satisfied while preserving statistical fidelity and downstream utility. Existing approaches provide subsets of these capabilities—for example, P-DGM refines black-box outputs under supplied linear constraints (Stoian et al., 2024). Our framework instead couples a shared executable discovery-and-revision interface across three families with coordinated post-hoc repair, without generator access or retraining.

3 Methodology

3.1 LLM-Guided Constraint Discovery

As summarized in Figure 1, our family-specific LLM workflow profiles the reference table, generates executable hypotheses, validates and revises them using deterministic tools, and consolidates

accepted constraints for subsequent enforcement.

Dataset profiling. We first construct compact column profiles of $\mathcal{D}_{\text{real}}$, similar to the practices in previous works involving LLMs in tabular data analysis (Rahman et al., 2026; Long et al., 2025; Akella et al., 2025). For each column, the profile contains its semantic description, data types (num/cat), and type-specific statistics. Numerical statistics include ranges, quantiles, means, stds, and mass points, while categorical statistics include frequent values and their frequencies.

Executable constraint hypothesis generation. For each constraint family, the discovery LLM agent receives the dataset description, column profiles, and a context set of $n_{\text{ctx}} \ll N$ reference rows sampled without replacement. Using a fixed cap keeps prompt length and inference cost bounded. The sampled rows are used only for hypothesis generation; all generated hypotheses are validated against the full table.

The agent emits each candidate as a structured hypothesis containing (1) a unique identifier, (2) a natural-language description, (3) the participating columns, and (4) a family-specific machine-executable representation. The executable field is central to our design: it converts a semantic proposal into a deterministically evaluable object and provides a common interface for proposal, full-table execution, diagnosis, and revision across all three constraint families. The description supports human inspection, but only the executable field is used for validation. Prior work has generated executable tabular-quality validators or directly parseable linear constraints (Akella et al., 2025; Afonja et al., 2026); our framework unifies these capabilities within a typed validation-and-revision workflow. Appendix A.1 gives concrete examples of all supported representations.

An equational candidate h uses executable Python to represent an arithmetic identity among numerical columns. The generated code defines a single `check(df)` function that evaluates the identity row by row and returns an index-aligned Boolean Series. Because equations may involve free-form arithmetic expressions and dataset-specific numerical tolerances, an executable Python checker is more expressive than a fixed-vocabulary domain-specific language (DSL) while remaining deterministically evaluable (Example A.1).

A linear-inequality candidate h uses a JSON-encoded DSL to represent the feasible half-space

$$h(\mathbf{x}) : \sum_{j \in S_h} a_{h,j} x_j \bowtie_h b_h, \qquad \bowtie_h \in \{\leq, \geq\},$$

where $S_h \subseteq \{1, \ldots, d\}$ indexes the participating numerical columns, $a_{h,j}$ is the coefficient of column j, and b_h is the boundary constant. The DSL stores the participating columns, coefficient map, comparison sense, and right-hand-side constant, following prior work on constraint extraction (Afonja et al., 2026). By explicitly encoding both the boundary and its feasible side, this representation supports deterministic parsing, evaluation, canonicalization, and joint projection (Example A.2).

An LD candidate h uses a unified JSON-encoded DSL to represent categorical admissibility. It specifies one or more determinant columns $\mathbf{X}_h = (X_{h,1}, \ldots, X_{h,k_h})$, one dependent column Y_h, and a value table T_h. Each entry of T_h associates a nonempty value set for every determinant column with a nonempty set of admissible dependent values. A row matches an entry when its value for every determinant column belongs to the corresponding set, and it satisfies the entry when its dependent value belongs to the associated admissible set. Values within a determinant set are alternatives, whereas sets across determinant columns are matched jointly. This representation enables deterministic equality and set-membership tests without imposing an arbitrary geometry on categorical codes (Examples A.3 and A.4).

Tool-grounded validation and revision. Every proposed hypothesis h is evaluated on the complete reference dataset by a deterministic, family-specific tool: `validate_equational`, `validate_linear`, or `validate_LD`. Let $\mathcal{A}_h \subseteq \mathcal{D}_{\text{real}}$ denote the rows to which h applies and $\mathcal{V}_h \subseteq \mathcal{A}_h$ the rows that violate it. All three validators compute $\text{VR}(h) = |\mathcal{V}_h|/|\mathcal{A}_h|$, and retain h only if $\text{VR}(h) \leq \tau_{\text{vio}}$. Equational and linear-inequality hypotheses apply to every row, so $\mathcal{A}_h = \mathcal{D}_{\text{real}}$. For an LD hypothesis, $\mathcal{A}_h$ contains only rows whose determinant values match at least one entry of its value table T_h; rows that match no entry are outside the hypothesis's scope.

Each validator computes $|\mathcal{V}_h|$ from the family-specific executable representation defined above. `validate_equational` executes the generated `check(df)` function and counts the `False` entries in its returned Boolean Series. `validate_linear`

evaluates the DSL-defined half-space and counts rows outside its feasible region. `validate_LD` counts applicable rows with dependent values outside their matched admissible sets.

LD hypotheses additionally require sufficient empirical support. Association-rule mining uses support to distinguish statistically meaningful rules from patterns observed too infrequently to merit consideration (Agrawal et al., 1993). Analogously, `validate_LD` calculates applicability support as $\mathrm{Supp}(h) = |\mathcal{A}_h|/N$ and retain an LD hypothesis only if $\mathrm{Supp}(h) \geq \tau_{\mathrm{sup}}$. This criterion complements the violation rate by measuring the breadth of evidence for the hypothesis, preventing a narrowly scoped rule from passing on too few applicable rows. We set both thresholds to 0.005 in all experiments, i.e., $\tau_{\mathrm{vio}} = \tau_{\mathrm{sup}} = 0.005$.

When h fails validation, the validator returns aggregate diagnostics and up to n_{cex} violating rows as counterexamples to the same discovery agent to request for a revision. The agent may make at most R_{rev} revision attempts after the initial validation, stopping early upon acceptance. A candidate that still fails after the final attempt is rejected.

Multi-round discovery. A single discovery LLM call may focus on the most salient relationships and overlook less obvious constraints. We therefore conduct R_{disc} discovery rounds per constraint family for every dataset. In each follow-up round, the agent receives a summary of the hypotheses accepted and rejected in previous rounds and is instructed to explore explore relationships that haven't been explored in previous discovery rounds. Unless otherwise stated, we use $n_{\mathrm{ctx}} = 100$ context rows, $n_{\mathrm{cex}} = 20$ counterexamples, $R_{\mathrm{rev}} = 3$ revision attempts, and $R_{\mathrm{disc}} = 3$ discovery rounds. This configuration balances broad constraint coverage with token usage, as shown by the ablation of n_{ctx}, n_{cex}, and R_{disc} in Appendix E.1.

Consolidation. After R_{disc} discovery rounds, we apply family-specific consolidation to remove redundant or structurally overlapping constraints. For linear inequalities, we first rewrite each candidate as $\mathbf{a}_h^\top \mathbf{x} \leq b_h$ and normalize it by a positive scale. We remove exact duplicates and, among inequalities with an identical normalized coefficient vector $\mathbf{a}_h$, retain only the smallest right-hand side b_h. For equational constraints, we first retain one preferred candidate for each participating-column set. We then solve a binary optimization problem that maximizes the number of retained equations subject to every retained equation having at least one participating column unused by every other retained equation. For LD constraints, we enforce an acyclic dependency graph in acceptance order, canonicalize value tables into atomic mappings, remove exact duplicates and strictly subsumed mappings, and merge compatible mappings with the same determinant–dependent signature into one value table representation.

3.2 Constraint Enforcement

We postprocess $\mathcal{D}_{\mathrm{syn}}$ without accessing or changing the backbone, making the same pipeline applicable to black-box generators and already-produced tables. This avoids the retraining or training-pipeline changes required by generator-integrated methods (Stoian et al., 2024; Vero et al., 2023; Afonja et al., 2026; Long et al., 2025). We also avoid rejection sampling: at high violation rates, acceptance becomes impractically low, while selective retention can distort the generated distribution and downstream utility (Stoian and Giunchiglia, 2025), which motivates direct post-hoc constraint repair.

We enforce logical dependencies, equations, and linear inequalities in sequence. Because logical repairs modify only categorical columns, their ordering relative to the numerical repair stages is immaterial. Equation repair, however, must strictly precede linear projection: reconstructing a column after projection can reintroduce inequality violations. During projection, we therefore hold equation-participating columns fixed, preserving the repaired equations throughout the final stage. Our ablation in Appendix E.3 shows that reversing the equation-repair and projection stages can reintroduce previously resolved violations.

Logical-dependency enforcement. For each violated LD, we keep the determinant fixed and replace only the dependent value with a value from the admissible set specified by its matching value-table entry. We assign singleton sets deterministically and otherwise sample from the reference conditional distribution restricted to the set. This satisfies the rule where a repair is defined and favors reference-like conditional frequencies rather than guaranteeing marginal preservation. For overlapping rules, we intersect admissible sets per dependent column set and topologically process the acyclic determinant-to-dependent graph. Omitted determinant configurations remain outside the

rule's scope.

Equational-constraint enforcement. A checker identifies violations but does not specify a repair. For each equation c and possible target j, an LLM agent generates $g_{c\to j}$ (implemented as `fix(df)`) to reconstruct j from the other participating columns. We validate it on the complete reference table using the protocol in Section 3.1 and retain it only if $\mathrm{VR}(c) \le \tau_{\text{vio}}$ under the frozen checker; otherwise, it is revised or discarded. For equations that share columns, the KS-complement-guided heuristic described in Appendix A.3 jointly selects the repair targets and their execution order from all feasible complete schedules. It first maximizes the worst predicted change in marginal KS complement, avoiding schedules that severely degrade any single column, and then selects the remaining schedule with the largest total predicted change. We use the KS complement because preserving marginal distributions is important for downstream statistical utility. Appendix E.2 evaluates this heuristic against randomized target and order selection constrained to retain a complete repair schedule. After each repair, participating columns are protected and resolved equations are revalidated.

Linear-inequality enforcement. We jointly project each violating row onto the common linear feasible region. Stoian et al. (2024) sequentially correct features using a specified variable order, whereas EVS uses order-free Euclidean projection (Afonja et al., 2026). However, both P-DGM and EVS enforce linear inequalities in isolation; neither method is able to coordinate the projection while preserving previously repaired equational constraints. Our joint convex program instead minimizes scale-normalized distortion while fixing columns in repaired equations, yielding an order-independent global optimum that preserves those repairs. Let $\widetilde{\mathbf{x}}_i$ be the row after the preceding stages, S_{num} its numerical-column indices, and $\mathbf{z} = (z_j)_{j\in S_{\text{num}}} \in \mathbb{R}^{|S_{\text{num}}|}$. Stack the canonicalized inequalities as $A\mathbf{z} \le \mathbf{b}$. Let $P \subseteq S_{\text{num}}$ contain the equation-participating and zero-variance columns, and let $U = S_{\text{num}} \setminus P$:

$$\widehat{\mathbf{x}}_{i,S_{\text{num}}} \in \underset{\mathbf{z}\in\mathbb{R}^{|S_{\text{num}}|}}{\arg\min} \sum_{j\in U} \left(\frac{z_j - \widetilde{x}_{ij}}{s_j}\right)^2$$
$$\text{subject to} \quad A\mathbf{z} \le \mathbf{b},$$
$$z_j = \widetilde{x}_{ij} \quad \forall j \in P.$$

For $j \in U$, $s_j > 0$ is the reference standard deviation. Integer-valued columns use their continuous relaxation; this stage does not enforce integrality.

Finally, after three stages of repairing, all family-specific validators recheck the postprocessed table to ensure the constraints are all satisfied.

4 Evaluation

4.1 Executable Constraint Discovery

We evaluate executable discovery on three curated public datasets, each representing a different constraint family: NBA contains 12 equations relating derived basketball statistics; URL contains 17 inequalities over numerical webpage features; and a categorical version of ANXIETY contains 13 logical dependencies (LDs) encoding implications and admissible mappings. Using separate benchmarks provides family-specific curated constraints and controlled audit tasks for each typed representation; Section 4.2 evaluates the full pipeline when multiple families are discovered and enforced together. For five 70/30 discovery–audit splits, both methods see only the discovery partition; hidden audit rows form balanced contrastive tasks for every constraint.

We access GPT-5.6 Luna (OpenAI, 2026) and Claude Sonnet 5 (Anthropic, 2026) through their APIs and use each as the LLM backbone for both methods. The baseline makes one direct-prompt call without verification tools or revision; ours runs the full tool-grounded agentic workflow from Section 3.1. We report held-out violation-detection accuracy, precision, and recall using a support-weighted macro-average across constraint-specific tasks. Appendix C summarizes the annotations and task construction.

Our complete executable validation-and-revision workflow achieves higher mean held-out violation-detection scores in all 18 family–backbone–metric comparisons. Table 1 shows the largest gains for LDs: precision and recall each rise from .442 to .974 with GPT-5.6 and .963 with Claude-5. Linear discovery also improves substantially, especially with Claude-5, whose accuracy increases from .815 to .959 and recall from .633 to .919. Even against near-ceiling equational baselines, our workflow reaches 1.000 on all three metrics with GPT-5.6 and .998 accuracy, 1.000 precision, and .996 recall with Claude-5. The simultaneous precision and recall gains indicate better discrimination between valid audit rows and con-

Type	Model	Method	Acc.	Prec.	Rec.
Linear	GPT-5.6	Baseline	.842 (.016)	.741 (.032)	.684 (.032)
		Ours	**.917** (.096)	**.911** (.181)	**.833** (.193)
	Claude-5	Baseline	.815 (.033)	.729 (.089)	.633 (.067)
		Ours	**.959** (.016)	**.975** (.031)	**.919** (.032)
Equational	GPT-5.6	Baseline	.975 (.037)	.950 (.075)	.950 (.075)
		Ours	**1.000** (.000)	**1.000** (.000)	**1.000** (.000)
	Claude-5	Baseline	.960 (.025)	.933 (.037)	.919 (.051)
		Ours	**.998** (.002)	**1.000** (.000)	**.996** (.003)
LD	GPT-5.6	Baseline	.721 (.000)	.442 (.000)	.442 (.000)
		Ours	**.987** (.016)	**.974** (.032)	**.974** (.032)
	Claude-5	Baseline	.721 (.000)	.442 (.000)	.442 (.000)
		Ours	**.982** (.007)	**.963** (.014)	**.963** (.014)

Table 1: Held-out behavioral violation-detection results over five discovery–audit splits (mean and standard deviation). The audit tasks are constructed from 17 inequalities, 12 equations, and 13 logical dependencies.

trastive violations; gains across both backbones suggest that the benefit is not model-specific.

4.2 End-to-End Constraint-Aware Generation

Experimental setup. We evaluate seven public datasets (Details in Appendix B, ANXIETY exluded since it's purely categorical and has no target column) using four generators that span distinct modeling paradigms: Gaussian Copula (statistical) (Patki et al., 2016), CTGAN (adversarial) and TVAE (variational autoencoding) (Xu et al., 2019), and TabDDPM (diffusion-based) (Kotelnikov et al., 2023). For each of three independent 70/30 splits, we train every generator once and draw three synthetic tables. Constraints are independently discovered from each real training partition using our agentic workflow with GPT-5.6 Luna backbone and default hyperparameters and applied to the corresponding synthetic outputs, yielding 252 paired raw–postprocessed evaluations. Across datasets and splits, the system discovers 0–16 LD constraints, 0–12 equational constraints, and 0–29 linear inequalities per dataset; Table 2 reports the dataset-specific counts and split-dependent ranges. Appendix D provides the full protocol and configuration details.

Metrics. CVR is the fraction of rows violating at least one constraint (perfect: CVR $= 0$), whereas sCVC measures the density of violations across all row–constraint pairs. Equational R^2 measures consistency with discovered equations (perfect: $R^2 = 1$), and linear feasibility distance (LFD) measures distance from the jointly feasible linear region (perfect: LFD $= 0$). We assess downstream utility using the Train on Synthetic, Test on Real (TSTR) protocol (Stoian et al., 2025). Each dataset table

	Logical dependencies CVR (%) ↓ · [0, 100]	Equations R^2 ↑ · $(-\infty, 1]$	Linear inequalities LFD ↓ · $[0, \infty)$
CTGAN	19.0 ± 31.8	WORST −0.97 ± 2.91	0.30 ± 0.38
Gaussian Copula	WORST 36.4 ± 33.9	0.12 ± 0.33	0.44 ± 0.89
TabDDPM	BEST 3.1 ± 4.4	BEST 0.45 ± 0.55	WORST 2.23 ± 2.95
TVAE	19.0 ± 27.2	BEST 0.45 ± 0.19	BEST 0.11 ± 0.12

Figure 2: Generator-wise raw constraint metrics (mean ± standard deviation across applicable datasets). Blue and vermillion cells mark the best and worst means, respectively, within each constraint family.

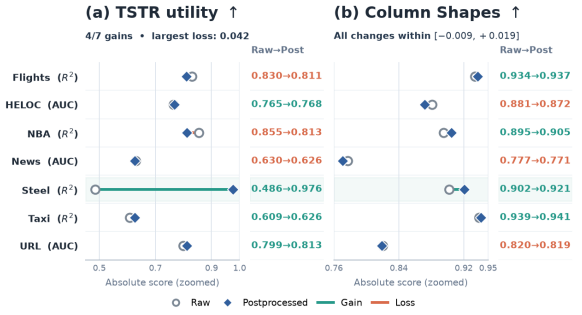

Figure 3: Raw-to-postprocessed TSTR utility and Column Shapes. TSTR uses ROC-AUC for classification datasets and R^2 otherwise; only within-dataset changes are comparable.

contains a target column: three define classification tasks, evaluated using ROC-AUC, and four define regression tasks, evaluated using R^2. We assess univariate marginal fidelity using Column Shapes. Figure 3 reports absolute raw and postprocessed scores; definitions are in Appendix D.

Raw generators exhibit distinct, constraint-specific failure profiles. Figure 2 reveals reversals in relative performance across families. TabDDPM has the lowest LD CVR (3.1%) but the highest LFD (2.23), whereas TVAE has the lowest LFD (0.11); both attain the highest equational R^2 (0.45) at the reported precision. Conversely, CTGAN is weakest on equations ($R^2 = -0.97$), and Gaussian Copula has the highest LD CVR (36.4%). Thus, preserving one family well does not predict preservation of another. Generator selection alone is therefore insufficient for structural validity, motivating family-aware enforcement that remains agnostic to generator architecture.

Postprocessing exactly satisfies every applicable family while largely preserving utility and fidelity. Table 2 shows substantial violations in raw outputs: equational CVR ranges from 98.2%

Metric		FLIGHTS	HELOC	NBA	NEWS	STEEL	TAXI	URL
Constraint # (LD/Eq/Lin)		3/7/2–5	1/0/11–13	0/10–12/3–11	7–16/4/0–17	1/2/11–25	2/2/0	2/0/17–29
LD	CVR (%) ↓	65.9 → **0**	4.0 → **0**	–	20.2 → **0**	11.7 → **0**	12.2 → **0**	2.2 → **0**
	sCVC (%) ↓	40.0 → **0**	4.0 → **0**	–	2.2 → **0**	11.7 → **0**	6.3 → **0**	1.1 → **0**
Eq.	CVR (%) ↓	100.0 → **0**	–	99.9 → **0**	100.0 → **0**	98.2 → **0**	100.0 → **0**	–
	sCVC (%) ↓	99.3 → **0**	–	94.4 → **0**	86.1 → **0**	63.7 → **0**	96.6 → **0**	–
	R^2 ↑	0.31 → **1**	–	−1.30 → **1**	−0.19 → **1**	0.65 → **1**	0.58 → **1**	–
Lin.	CVR (%) ↓	0.0 → **0**	58.7 → **0**	8.0 → **0**	42.3 → **0**	5.7 → **0**	–	77.8 → **0**
	sCVC (%) ↓	0.0 → **0**	7.9 → **0**	0.7 → **0**	5.7 → **0**	0.2 → **0**	–	11.1 → **0**
	LFD ↓	0.000 → **0**	1.169 → **0**	0.029 → **0**	1.483 → **0**	0.020 → **0**	–	1.912 → **0**

Table 2: End-to-end constraint enforcement across seven public datasets. Constraint-metric cells show raw → postprocessed means; exact optima are written as 0 or 1. Dataset means average samples within each generator and split, then generators and splits equally. Constraint counts span splits; dashes denote unavailable families.

to 100%, LD CVR reaches 65.9%, and linear CVR reaches 77.8%. Postprocessing nevertheless reaches the exact optimum for every applicable metric: CVR, sCVC, and LFD fall to zero, while equational consistency rises to $R^2 = 1$. Reaching these targets jointly across heterogeneous family combinations shows that repair does not merely shift violations from one constraint type to another. Figure 3 grounds these changes in their absolute baselines: mean utility improves on four of seven datasets (STEEL: 0.486 → 0.976) and decreases by at most 0.042, while Column Shapes changes by only −0.009 to +0.019. Thus, exact enforcement largely preserves univariate marginal fidelity and can sometimes benefit downstream prediction.

4.3 Ablation Studies

Discovery configuration. Counterexamples and repeated discovery increase retained yield, but larger budgets show diminishing returns. On NEWS (Figure 4), removing counterexamples and using one rather than three rounds lower mean validated-hypothesis yield from 46.33 to 19.33 and 11.33, respectively. Five rounds more than double token use but yield only 38.67; alternative context and larger counterexample budgets offer at most marginal gains. Thus, the default $(3, 100, 20)$ is the best observed cost–yield trade-off among the tested settings, not a universal optimum (Appendix E.1).

Equational repair scheduling. At equal constraint satisfaction, KS-complement guidance has higher mean marginal fidelity than random feasible scheduling. Both attain zero equational CVR on 180 matched inputs from five datasets. With inputs, constraints, and repair functions fixed, KS guidance has higher mean paired Column Shapes on four datasets by 0.020–0.053, ties on NEWS, and yields an equal-weight mean gain of 0.025 (Appendix E.2).

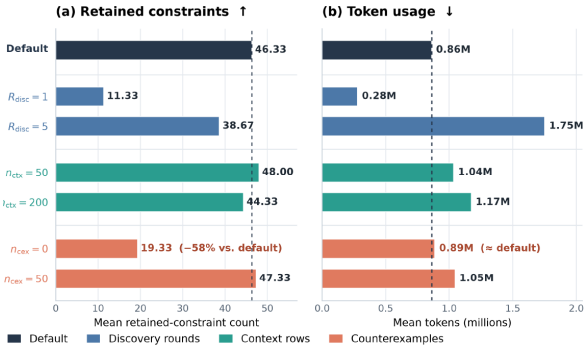

Figure 4: Discovery sensitivity on NEWS (three-split means; GPT-5.6 Luna). Each variant changes one component of the default $(3, 100, 20)$; dashed lines mark it. Higher yield and lower token use are better.

Cross-family repair order. Projection last avoids observed cross-family interference. On NBA, with E denoting equational repair and L linear-inequality projection, $E \to L$ yields zero final CVR for both families and zero LFD in all 36 matched settings. The reverse leaves linear violations in 24 settings (66.7%; CVR 4.574%; LFD 0.863), while equational CVR remains zero. This supports projection last when numerical families overlap (Appendix E.3).

5 Conclusion

We presented a unified framework that turns LLM-proposed equations, linear inequalities, and logical dependencies into executable hypotheses and coordinates their post-hoc enforcement on outputs from unchanged generators. Across multiple LLMs, datasets, and generator families, the complete validation-and-revision workflow improves held-out violation detection over direct prompting, while postprocessing yields zero measured viola-

tions under retained, applicable validators with generally preserved utility and univariate marginal fidelity. Coupling counterexample-grounded discovery with coordinated repair offers a practical path to structurally reliable tabular synthesis.

Limitations

Our method is limited to inter-column constraints evaluated within each record. It does not model inter-row constraints, such as uniqueness across records, or cell-level validity and formatting rules, such as regular-expression patterns and domain-specific semantic types supported by SDV (DataCebo, 2026). Extending discovery and enforcement to these constraint families is outside the scope of this work.

Full-table validation establishes empirical consistency, not semantic correctness or complete discovery. Zero measured violations therefore applies only to retained, applicable validators and excludes uncovered LD configurations or unavailable repairs.

In all experiments, we supply rich metadata, including dataset- and column-level descriptions, which the LLM discovery agents use to propose constraint hypotheses. In practice, however, many tables may lack such annotations or have ambiguous and noisy column semantics, potentially reducing discovery accuracy. Evaluating robustness to limited-quality metadata remains future work.

Ethics Statement

We use public tabular datasets. Some datasets nevertheless represent potentially sensitive domains, including credit risk and mental health. We use these datasets only for methodological evaluation and do not attempt to identify individuals or support decisions about them.

The LLM components receive dataset metadata, column profiles, sampled records, and verifier-selected counterexamples. Consequently, applying the framework to confidential or personally identifiable data would require appropriate authorization, data-protection measures, and careful consideration before transmitting records to third-party model providers.

Structural validity does not guarantee privacy or fairness: discovered constraints may encode historical biases and require domain review. Generated Python must be securely sandboxed.

References

Tejumade Afonja, Joscha Cüppers, and Mario Fritz. 2026. From noisy oracles to useful constraints: LLM-guided constraint selection for synthetic tabular data. In *2nd ICML Workshop on Foundation Models for Structured Data*.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, New York, NY, USA. Association for Computing Machinery.

Ashlesha Akella, Akshar Kaul, Krishnasuri Narayanam, and Sameep Mehta. 2025. Quality assessment of tabular data using large language models and code generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2713–2748, Suzhou (China). Association for Computational Linguistics.

Anthropic. 2026. Claude Sonnet 5. Anthropic model documentation. Accessed 2026-07-30.

Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. 2023. Language models are realistic tabular data generators. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

DataCebo. 2026. Synthetic data vault documentation. Online documentation. Accessed 2026-08-01.

Mary Grace Dhooghe, Murat Kantarcioglu, and Bhavani Thuraisingham. 2026. MoE-T: Dependency graph-gated mixture of experts for tabular generation with functional dependency preservation. In *Proceedings of the Sixteenth ACM Conference on Data and Application Security and Privacy*.

Jayoung Kim, Chaejeong Lee, and Noseong Park. 2023. Stasy: Score-based tabular data synthesis. In *11th International Conference on Learning Representations, ICLR 2023*.

Jinhee Kim, Taesung Kim, and Jaegul Choo. 2024. EPIC: effective prompting for imbalanced-class data synthesis in tabular data classification via large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. 2023. Tabddpm: Modelling tabular data with diffusion models. In *International conference on machine learning*, pages 17564–17579. PMLR.

Siyi Liu, Yujia Zheng, and Yongqi Zhang. 2025. Structsynth: Leveraging llms for structure-aware tabular data synthesis in low-data regimes. *arXiv preprint arXiv:2508.02601*.

Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. 2023. Goggle: Generative modelling for tabular data by learning relational structure. In *The Eleventh International Conference on Learning Representations*.

Yunbo Long, Liming Xu, and Alexandra Brintrup. 2025. Llm-tablogic: Preserving inter-column logical relationships in synthetic tabular data via prompt-guided latent diffusion. *arXiv preprint arXiv:2503.02161*.

OpenAI. 2026. GPT-5.6 Luna Model. OpenAI API documentation. Accessed 2026-07-30.

Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10).

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. 2016. The synthetic data vault. In *2016 IEEE international conference on data science and advanced analytics (DSAA)*, pages 399–410. IEEE.

Adil Rahman, Koichiro Niinuma, and Aakar Gupta. 2026. Dataspeck: An ai-driven human-in-the-loop system for automating transformations in data conversion workflows. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI '26, New York, NY, USA. Association for Computing Machinery.

Nabeel Seedat, Nicolas Huynh, Boris van Breugel, and Mihaela van der Schaar. 2024. Curated LLM: synergy of llms and data curation for tabular augmentation in low-data regimes. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 44060–44092. PMLR / OpenReview.net.

Ruxue Shi, Yili Wang, Mengnan Du, Xu Shen, Yi Chang, and Xin Wang. 2025. A comprehensive survey of synthetic tabular data generation. *arXiv preprint arXiv:2504.16506*.

Mihaela Stoian and Eleonora Giunchiglia. 2025. Beyond the convexity assumption: Realistic tabular data generation under quantifier-free real linear constraints. In *International Conference on Learning Representations*, volume 2025, pages 28812–28841.

Mihaela C Stoian, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. 2024. How realistic is your synthetic data? constraining deep generative models for tabular data. In *The Twelfth International Conference on Learning Representations*.

Mihaela C Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. 2025. A survey on deep learning approaches for tabular data generation: Utility, alignment, fidelity, privacy, diversity, and beyond. *Transactions on Machine Learning Research*.

Chaithra Umesh, Kristian Schultz, Manjunath Mahendra, Saptarshi Bej, and Olaf Wolkenhauer. 2025. Preserving logical and functional dependencies in synthetic tabular data. *Pattern Recognition*, 163:111459.

Chaithra Umesh, Kristian Seegel-Schultz, Manjunath Mahendra, Saptarshi Bej, and Olaf Wolkenhauer. 2026. Dependency-aware synthetic tabular data generation. *Pattern Recognition*, 179:113819.

Mark Vero, Mislav Balunović, and Martin Vechev. 2023. Cuts: Customizable tabular synthetic data generation. *arXiv preprint arXiv:2307.03577*.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32.

Mingming Zhang, Zhiqing Xiao, Guoshan Lu, Sai Wu, Weiqiang Wang, Xing Fu, Can Yi, and Junbo Zhao. 2025. AIGT: AI generative table based on prompt. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9926–9938, Abu Dhabi, UAE. Association for Computational Linguistics.

Tianping Zhang, Shaowen Wang, Shuicheng Yan, Li Jian, and Qian Liu. 2023. Generative table pretraining empowers models for tabular prediction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 14836–14854. Association for Computational Linguistics.

A Constraint Discovery and Enforcement Details

A.1 Constraint Hypothesis Representations

The discovery agent emits a different machine-executable representation for each constraint family. The examples below show the concrete schemas referenced in Section 3.1. We use a display-oriented rendering for readability; in the equational example, the escaped line breaks stored in the check_code string are expanded into an indented Python block.

Example A.1: Equational

```
id:
  "eq_absolute_title_subjectivity"

description:
  "abs_title_subjectivity is the absolute
   distance of title_subjectivity from 0.5."

columns:
  ["abs_title_subjectivity", "title_subjectivity"]

check_code:
def check(df):
    absolute = pd.to_numeric(
        df["abs_title_subjectivity"],
        errors="coerce",
    )
    subjectivity = pd.to_numeric(
        df["title_subjectivity"],
        errors="coerce",
    )
    finite = (
        absolute.notna()
        & subjectivity.notna()
        & np.isfinite(absolute)
        & np.isfinite(subjectivity)
    )
    error = absolute - (subjectivity - 0.5).abs()
    return finite & error.abs().le(1e-9)
```

Example A.2: Linear inequality

```
{
"id": "avg_negative_polarity_ge_min",
"description": "Average negative-word polarity is at least
      the minimum negative-word polarity.",
"columns": [
  "avg_negative_polarity",
  "min_negative_polarity"
],
"coefficients": {
  "avg_negative_polarity": 1,
  "min_negative_polarity": -1
},
"sense": ">=",
"rhs": 0
}
```

Example A.3: Logical dependency

```
{
  "id": "plan_severity_support_channel",
  "description": "The subscription plan and issue severity
     jointly restrict the available support channels.",
  "determinants": [
    "subscription_plan",
    "issue_severity"
  ],
  "dependent": "support_channel",
  "value_table": [
    {
      "determinant_values": [
        ["free", "standard"],  # IF plan: free OR standard
        ["low"]                #  AND severity: low
      ],
      "dependent_values": [    # THEN channel is one of
        "community_forum",
        "email"
      ]
    },
    {
      "determinant_values": [
        ["premium"],        # IF plan: premium
        ["low", "medium"]   #  AND severity: low OR medium
      ],
      "dependent_values": [ # Then channel is one of
        "email",
        "live_chat"
      ]
    },
    {
      "determinant_values": [
        ["premium", "enterprise"], # IF plan: premium OR
      enterprise
        ["high", "critical"]       #  AND severity: high OR
       critical
      ],
      "dependent_values": [        # Then channel is one of
        "priority_email",
        "live_chat",
        "phone"
      ]
    }
  ]
}
```

Example A.4: Logical dependency

```
{
  "id": "cat_fd_001",
  "description": "Each airport code determines its city.",
  "determinants": ["airport_code"],
  "dependent": "city",
  "value_table": [
    {
      "determinant_values": [["JFK"]], # IF airport_code:
      JFK
      "dependent_values": ["New York"] # THEN city: New
      York
    },
    {
      "determinant_values": [["SFO"]],
      "dependent_values": ["San Francisco"]
    },
    {
      "determinant_values": [["SEA"]],
      "dependent_values": ["Seattle"]
    },
    ...
  ]
}
```

A.2 Discovery LLM Prompts

We use a family-specific prompt for each constraint representation. The templates below show the instructions supplied to the discovery agent; braces denote values inserted by the host at run time.

Equational constraint discovery prompt

```
[System]
You are a meticulous tabular-data semantics auditor. Identify domain-defensible, deterministic numerical
equations between columns using the dataset description, column metadata, units, representative records, and
full-data verification. Prefer correctness, simplicity, and generality over the number of constraints.

A valid constraint expresses one row-wise numerical relationship that should hold for essentially every row.
Exclude correlations, trends, inequalities, distributional or conditional rules, and formulas fitted to the
sample.

For each candidate, provide exactly:
1. id: unique snake_case identifier;
2. description: concise statement of the equation;
3. columns: all and only referenced numerical columns; and
4. check_code: vectorized check(df) returning an index-aligned Boolean pandas Series, with True for satisfying
rows.

The checker must not import, mutate df, perform I/O, access external state, use row indices, hard-code
examples or exceptions, fit parameters, or use full-dataset aggregates.
Use exact comparisons for integer identities and only a small, semantically justified tolerance for
floating-point identities.

[User]
Dataset description:
{dataset_description}

Column profiles:
{numerical_column_profiles}

Data sample:
{sampled_rows}

Already accepted hypotheses (do not repeat or rearrange):
{accepted_hypotheses_json}

Previously rejected hypotheses (do not repeat or revise):
{rejected_hypotheses_json}

Discover as many non-speculative constraints as the evidence supports and submit them together in one verifier
call.
```

Logical dependency discovery prompt

```
[System]
You are a meticulous tabular-data semantics auditor. Discover semantically meaningful logical dependencies
among categorical columns using column metadata, representative records, and full-data evidence.

A constraint has one or more determinant columns, exactly one dependent column, and a value table. Each table
entry lists admissible values for every determinant and for the dependent. A row is applicable when its
determinant values match an entry and violates the constraint when its dependent value is not admissible;
unmatched rows do not violate it.

For a large exact dependency, you may submit an empty value table; the host then builds the complete mapping
using the majority dependent value for each observed determinant configuration. Conditional and
multi-admissible mappings require an explicit value table.

For each candidate, provide exactly:
1. id: unique snake_case identifier;
2. description: concise statement of the dependency;
3. columns: all and only referenced categorical columns; and
4. constraint_representation: {logical_dependency_DSL_format}


Prioritize semantically grounded relationships such as codes determining names, geographic identifiers
determining regions, classifications determining labels, and stable status lookups.
Prefer minimal determinant sets. Avoid row keys, identifiers, nearly unique determinants, accidental sample
patterns, tautological tables, and arbitrary groupings created only to reduce violations.


[User]
Dataset description:
{dataset_description}

Column profiles:
{categorical_column_profiles}

Data sample:
{sampled_rows}

Already accepted hypotheses (do not repeat or rearrange):
{accepted_hypotheses_json}

Previously rejected hypotheses (do not repeat or revise):
{rejected_hypotheses_json}

Inspect and analyze promising logical dependencies. Finish with the required structured completion when no
additional semantically defensible logical dependency remains.
```

Linear-inequality discovery prompt

```
[System]
You are a meticulous tabular-data semantics auditor. Discover semantically meaningful, row-wise linear
inequalities among numerical columns.

A useful constraint is a universal semantic relationship expected to hold for essentially every row.
Prioritize simple inequalities justified by column definitions, especially totals covering subsets,
minimum--average--maximum orderings, lengths or capacities covering components, and other whole--part
relationships.

For each candidate, provide exactly:
1. id: unique snake_case identifier;
2. description: concise statement of the linear inequality;
3. columns: all and only referenced numerical columns; and
4. constraint_representation: {linear_DSL_format}

Use small integer coefficients when possible. Exclude correlations, trends, fitted regressions, quantile or
distributional claims, conditional rules, unlisted columns, and arbitrary constants selected from sample
extrema.  Do not submit positive scalar multiples of an existing inequality, or repeat, weaken, or rescale an
accepted or rejected hypothesis.

[User]

Dataset description:
{dataset_description}

Column profiles:
{numerical_column_profiles}

Data sample:
{sampled_rows}

Already accepted hypotheses (do not repeat or rearrange):
{accepted_hypotheses_json}

Previously rejected hypotheses (do not repeat or revise):
{rejected_hypotheses_json}

Discover all distinct, semantically defensible linear inequalities supported by the evidence. Avoid previous
hypotheses and scaled rewrites.
```

Constraint refinement prompt

```
[User]
Refinement round {refinement_round} of {max_refinement_rounds} for discovery phase {phase}.

Column profiles:
{involved_column_profiles}

Samples that failed at validation during the previous round:
{counterfactual_samples}

Candidate and full verification history, including violating samples from every previous attempt:
{candidate_history_json}

Submit exactly one revision with the same constraint ID. If no principled revision remains, finish with
`rejected_hypotheses` listing that ID and a concrete reason.
```

A.3 KS-Guided Equational Repair

Given a synthetic table S, a real training table R, and a set of equational constraints $\mathcal{C}$, we need to choose which column to repair for each constraint and the order in which to perform the repairs. Different choices may satisfy the same constraints but have different effects on the quality of the synthetic data. We measure the similarity between a real column R_j and a synthetic column S_j using the KS complement:

$$Q(R_j, S_j) = 1 - D_{\mathrm{KS}}(R_j, S_j),$$

where a larger value indicates more similar marginal distributions.

For each constraint c, let $\mathcal{T}_c$ be the set of columns that can be repaired. For each possible target $j \in \mathcal{T}_c$, we apply the repair $f_{c,j}$ to the original synthetic table and compute

$$\Delta_{c,j} = Q(R_j, f_{c,j}(S)_j) - Q(R_j, S_j).$$

The value $\Delta_{c,j}$ estimates how repairing column j changes its marginal similarity to the real data. A positive value indicates an expected improvement, while a negative value indicates an expected loss.

Repair order matters because different constraints may share columns. Once a constraint is repaired, we freeze all columns involved in that constraint. Later repairs may use these columns as inputs but cannot overwrite them. This ensures that a later repair does not invalidate a constraint that has already been satisfied.

A repair schedule can be viewed as a path containing one target choice for each constraint. Among all valid paths, we choose the one whose lowest $\Delta_{c,j}$ is as high as possible:

$$\tau^\star = \max_{P \in \mathcal{P}} \min_{(c,j) \in P} \Delta_{c,j},$$

where $\mathcal{P}$ is the set of valid complete repair schedules. In other words, we seek the path with the best worst repair. This avoids choosing a schedule that contains one highly damaging repair, even if its other repairs have large improvements.

If several paths have the same best minimum value, we choose the one with the largest total score:

$$P^\star \in \arg \max_{\substack{P \in \mathcal{P}: \\ \min_{(c,j) \in P} \Delta_{c,j} \geq \tau^\star}} \sum_{(c,j) \in P} \Delta_{c,j}.$$

The first objective protects against a poor individual repair, while the second selects the path with the best overall predicted effect.

We solve this optimization problem by considering the distinct candidate scores $\Delta_{c,j}$ in descending order. For each score threshold τ, we use dynamic programming to search for a complete dependency-safe schedule using only repairs with $\Delta_{c,j} \geq \tau$. Each state records the set of constraints already scheduled, which determines the columns that are frozen and the repairs that remain available. The first threshold for which a complete schedule exists gives the optimal minimum score $\tau^\star$. Among all complete schedules satisfying this threshold, the dynamic program returns the one with the largest sum of candidate scores. The selected repairs are then applied sequentially in the resulting order.

B Datasets

For the eight public tabular datasets used in our experiments, we document their brief descriptions, sources, and tasks in downstream utility evaluations. Table 3 summarizes their Total-Train-Test sizes and categorical and numerical column counts.

Dataset	Total	Train	Test	Cat.	Num.
FLIGHTS	60,000	42,000	18,000	9	14
HELOC	9,084	6,358	2,726	3	21
NBA	14,062	9,843	4,219	0	26
NEWS	38,458	26,920	11,538	9	45
STEEL	34,993	24,495	10,498	3	7
TAXI	59,996	41,997	17,999	7	15
URL	11,430	8,001	3,429	50	37
ANXIETY	11,000	-	-	20	0

Table 3: Dataset sizes and column-type counts for the frozen end-to-end evaluation splits. Column counts include the utility target. Anxiety is not included in the end-to-end evaluation task.

Flights. We use a uniformly selected 60,000-row processed sample of completed U.S. flights from 2019–2023, sourced from the Kaggle Flight Delay and Cancellation dataset.[2] We remove cancelled or diverted flights; retain only rows whose origin and destination airports and cities are among the 100 most frequent values; and remove rows with missing or invalid clock-time or time-zone values. We then convert local flight times to UTC minute-of-day values. Its columns describe airlines, airports, route distance, scheduled and actual UTC clock times, taxi and air durations, and delays. The utility task is regression on `ARR_DELAY`.

[2] https://www.kaggle.com/datasets/patrickzel/flight-delay-and-cancellation-dataset-2019-2023

HELOC. The anonymized FICO Home Equity Line of Credit dataset is obtained through Hugging Face.[3] Each row represents a homeowner's credit application, with 23 credit-bureau attributes used to predict repayment performance over two years. The utility task is binary classification of `is_at_risk`: 1 denotes "Bad" performance and 0 denotes "Good." During preprocessing, we remove 588 rows in which all 23 predictors equal the missing-value code -9. We also remove 787 internally inconsistent rows in which `nr_total_trades` is smaller than at least one of its four component counts: `number_of_satisfactory_trades`, `nr_trades_initiated_in_last_year`, `nr_revolving_trades_with_balance`, or `nr_installment_trades_with_balance`.

NBA. The NBA data come from the pbpstats totals API.[4] They contain player-season scoring statistics for 30 regular seasons, from 1996–97 through 2025–26. Counts are normalized per 100 offensive possessions and describe shooting volume, efficiency, assisted scoring, putbacks, blocked attempts, and related measures. We replace missing numerical values—mostly sparse zero-count statistics—with zero. The utility task is regression on `Usage`, the percentage of team possessions ending in the player's shot, free-throw trip, or turnover.

News. The UCI Online News Popularity dataset contains feature summaries for Mashable articles published from 2013–2015.[5] Features cover article structure, keywords, channels, topics, links, and sentiment. We remove 1,186 rows in which both the positive- and negative-word rates are zero. We create the binary target `is_popular`, assigning 1 to articles with at least 1,400 shares and 0 otherwise, and then remove `shares` and the article URL. The utility task is binary classification of `is_popular`.

Steel. The Steel Industry Energy Consumption dataset records electricity use at 15-minute intervals over one calendar year.[6] Its variables cover reactive power, power factors, CO_2 emissions, time of day, weekday/weekend status, and operational load type. We remove 47 anomalous rows from the continuous interval on January 2, 2018, from 08:15 to 20:30. The interval contains 50 records; in the 47 removed records, `Usage_kWh` implies an expected CO_2 value between 0.01 and 0.07, whereas the recorded CO_2 value remains zero. The utility task is regression on `Usage_kWh`, the active electricity consumption.

Taxi. The taxi data are a processed sample of 2015 NYC Green Taxi trips from the NYC Taxi and Limousine Commission.[7] We remove the row identifier, `VendorID`, the original pickup and dropoff timestamps, and `Ehail_fee`, which is entirely missing. Columns describe locations, distance, passenger and fare codes, clock times, duration, and itemized charges. The utility task is regression on `total_amount`, which is the sum of fare, tax, tip, toll, extra, and improvement-surcharge of a trip.

URL. The Web Page Phishing Detection dataset is an exactly balanced benchmark with 5,715 legitimate and 5,715 phishing pages.[8] It combines URL syntax, fetched-page HTML content, and external-service features collected in May 2020. We remove the source `url` field because it is nearly unique, as well as `submit_email`. The utility task is binary classification of `status` as `legitimate` or `phishing`.

Anxiety. The Social Anxiety Dataset is an 11,000-row survey of anxiety severity and its lifestyle, physiological, and clinical correlates.[9] Columns describe demographics, lifestyle habits, family history and recent life events, physiological readings, treatment use, and self-reported stress and anxiety scores. It is used for constraint discovery only, as our logical-dependency benchmark, and has no utility task. Because logical dependencies are categorical, we discretize its twelve numerical columns into two to four ordinal bands each and derive `Occupation Group` from `Occupation`, giving a view of 11,000 rows and 20 categorical columns.

C Constraint Discovery Details

C.1 Expert Annotation and Ground-Truth Constraints

We construct the ground truth through a two-annotator protocol. The first annotator proposes

[3] https://huggingface.co/datasets/mstz/heloc

[4] https://api.pbpstats.com/docs

[5] https://archive.ics.uci.edu/dataset/332/online+news+popularity

[6] https://www.kaggle.com/datasets/csafrit2/steel-industry-energy-consumption

[7] https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[8] https://doi.org/10.17632/c2gw7fy2j4.2

[9] https://www.kaggle.com/datasets/natezhang123/social-anxiety-dataset

candidate constraints from the dataset documentation, schema, feature definitions, and domain semantics. The second annotator independently reviews every candidate along three dimensions: (i) *soundness*, whether the relation is satisfied by nearly all applicable records; (ii) *support*, whether the relation applies to enough records to be nontrivial—for categorical logical dependencies, each determinant configuration represented by a rule must occur in sufficiently many rows; and (iii) *semantic meaningfulness*, whether the constraint captures a substantive cross-column relation rather than an accidental regularity. We retain a candidate only if it satisfies all three criteria. This process yields 42 ground-truth constraints: 13 logical dependencies for ANXIETY, 12 equations for NBA, and 17 linear inequalities for URL. Tables 4–6 list the complete sets. These annotations remain hidden from all discovery methods and are used only for evaluation.

ANXIETY: logical dependencies. We use ANXIETY for logical-dependency evaluation because its low-cardinality categorical view contains semantically ordered demographic, lifestyle, physiological, and clinical bands. These properties naturally give rise to conditional admissibility rules—for example, a particular anxiety band restricting the allowed values of another band—and to mappings between original and derived categorical attributes.

The ANXIETY rules are meaningful because they encode admissible categories for the subgroup with the highest anxiety score, rather than claiming that the same associations hold for every participant.

See Table 4 for the full list of annotated logical dependency ground truths. The first eight high-anxiety rules categoricalize prior expert annotations; the sleep, stress, and sweating candidates were retained only after independent semantic review and zero-violation held-out validation; and the final two rules encode the one-to-many implications among two caffeine consumption-related columns.

NBA: equational constraints. We use NBA for equational-constraint evaluation because its numerical columns are per-100-possession basketball statistics, many of which are deterministically derived from shared event counts. Points, makes, attempts, assisted scoring, and percentages must therefore satisfy exact accounting identities rather than merely exhibit strong correlations. The annotators proposed and verified these rules with reference to the NBA Stats glossary for standard basketball terminology and statistical definitions.[10]

See Table 5 for the full list of annotated equations. In the column names, `FG` denotes field goal; `2` and `3` distinguish two- and three-point shots; `M` and `A` denote made and attempted; `Pts` denotes points; and `Pct` denotes a percentage represented as a fraction on a zero-to-one scale. All counting statistics are normalized per 100 offensive possessions. Equations 1-3 equations conserve points by decomposing them by shot value and assisted status; Equations 4–10 reconstruct makes, attempt shares, effective field-goal percentage, and assisted scoring from their defining numerators and denominators; and Equations 11–12 reconstruct blocked-attempt counts from their rates. Changing one participating value while holding the others fixed thus creates an internally inconsistent player-season record. We require finite numeric values and test equality with absolute and relative tolerances of 10^{-9}, which prevents floating-point representation from turning an otherwise valid identity into a violation.

URL: linear inequalities. We use URL for linear-inequality evaluation because its numerical features include nested string lengths, delimiter and resource counts, token order statistics, and complementary ratios. Containment, ordering, and bounded totals impose natural linear upper and lower bounds on these quantities, making the dataset especially well suited to semantically verifiable inequalities.

See Table 6 for the full list of annotated linear inequalities.

In the feature names, `length_url` and `length_hostname` denote character lengths, whereas `length_words_raw` counts raw tokens; `nb` denotes a count; `host` and `path` identify URL components; `raw` refers to tokens from the complete URL; `shortest`, `avg`, and `longest` summarize token lengths; and `int` and `ext` denote internal and external resources. Hyperlink ratios are fractions on a zero-to-one scale, whereas media ratios are percentages on a zero-to-100 scale. Feature names follow the source schema; `nb_semicolumn` is its original spelling. A component, token, or count cannot exceed the URL or hostname that contains it (Constraints 1–8 and 17), and the minimum, mean, and maximum token lengths must occur in that order (Constraints 9–14). Internal and external hyperlinks form disjoint

[10] https://www.nba.com/stats/help/glossary

#	Ground-truth logical dependency
1	Anxiety Band = Very high (9–10) → Caffeine Band = 294 mg or more
2	Anxiety Band = Very high (9–10) → Breathing Rate Band = 20 breaths/min or higher
3	Anxiety Band = Very high (9–10) → Heart Rate Band = 85 bpm or higher
4	Anxiety Band = Very high (9–10) → Alcohol Band = 5 or more drinks
5	Anxiety Band = Very high (9–10) → Diet Quality Band = Low (1–4)
6	Anxiety Band = Very high (9–10) → Therapy Band = 3 or more sessions
7	Anxiety Band = Very high (9–10) → Age Band = 20–49
8	Anxiety Band = Very high (9–10) → Physical Activity Band = 0–4 hours
9	Anxiety Band = Very high (9–10) → Sleep Band = {Under 6 hours, 6–9 hours}
10	Anxiety Band = Very high (9–10) → Stress Band = {High (7–8), Very high (9–10)}
11	Anxiety Band = Very high (9–10) → Sweating Band = {Moderate (3), High (4–5)}
12	Occupation = {Scientist, Doctor, Engineer, Lawyer} → Occupation Group = Higher caffeine floor; Occupation = {Student, Nurse, Freelancer, Chef} → Occupation Group = Moderate caffeine floor; Occupation = {Artist, Athlete, Musician, Other, Teacher} → Occupation Group = No annotated caffeine floor
13	Occupation Group = Higher caffeine floor → Caffeine Band = {250–293 mg, 294 mg or more}; Occupation Group = Moderate caffeine floor → Caffeine Band = {100–249 mg, 250–293 mg, 294 mg or more}

Table 4: The 13 annotator-verified logical dependencies used as ground truth for the ANXIETY detection task.

#	Ground-truth equation
1	$\texttt{Points} = 2 \times \texttt{FG2M} + 3 \times \texttt{FG3M} + \texttt{FtPoints}$
2	$2 \times \texttt{FG2M} = \texttt{PtsAssisted2s} + \texttt{PtsUnassisted2s}$
3	$3 \times \texttt{FG3M} = \texttt{PtsAssisted3s} + \texttt{PtsUnassisted3s}$
4	$\texttt{Fg2Pct} \times \texttt{FG2A} = \texttt{FG2M}$
5	$\texttt{Fg3Pct} \times \texttt{FG3A} = \texttt{FG3M}$
6	$\texttt{FG3APct} \times (\texttt{FG2A} + \texttt{FG3A}) = \texttt{FG3A}$
7	$\texttt{EfgPct} \times (\texttt{FG2A} + \texttt{FG3A}) = \texttt{FG2M} + 1.5 \times \texttt{FG3M}$
8	$\texttt{Assisted2sPct} \times (2 \times \texttt{FG2M}) = \texttt{PtsAssisted2s}$
9	$\texttt{Assisted3sPct} \times (3 \times \texttt{FG3M}) = \texttt{PtsAssisted3s}$
10	$\texttt{NonPutbacksAssisted2sPct} \times (2 \times \texttt{FG2M} - \texttt{PtsPutbacks}) = \texttt{PtsAssisted2s}$
11	$\texttt{FG2APctBlocked} \times \texttt{FG2A} = \texttt{Fg2aBlocked}$
12	$\texttt{FG3APctBlocked} \times \texttt{FG3A} = \texttt{Fg3aBlocked}$

Table 5: The 12 annotator-verified equations used as ground truth for the NBA detection task.

#	Ground-truth linear inequality
1	$\texttt{length_url} - \texttt{length_hostname} - \texttt{longest_word_path} - \texttt{nb_slash} \geq 0$
2	$\texttt{length_url} - \texttt{nb_dots} - \texttt{nb_hyphens} - \texttt{nb_and} - \texttt{nb_eq} - \texttt{nb_underscore} - \texttt{nb_percent} - \texttt{nb_slash} - \texttt{nb_semicolumn} \geq 0$
3	$\texttt{length_url} - \texttt{length_words_raw} \geq 0$
4	$\texttt{length_url} - \texttt{longest_words_raw} \geq 0$
5	$\texttt{length_hostname} - \texttt{longest_word_host} \geq 0$
6	$\texttt{longest_words_raw} - \texttt{longest_word_host} \geq 0$
7	$\texttt{longest_words_raw} - \texttt{longest_word_path} \geq 0$
8	$\texttt{shortest_word_host} - \texttt{shortest_words_raw} \geq 0$
9	$\texttt{longest_words_raw} - \texttt{avg_words_raw} \geq 0$
10	$\texttt{avg_words_raw} - \texttt{shortest_words_raw} \geq 0$
11	$\texttt{longest_word_host} - \texttt{avg_word_host} \geq 0$
12	$\texttt{avg_word_host} - \texttt{shortest_word_host} \geq 0$
13	$\texttt{longest_word_path} - \texttt{avg_word_path} \geq 0$
14	$\texttt{avg_word_path} - \texttt{shortest_word_path} \geq 0$
15	$-\texttt{ratio_intHyperlinks} - \texttt{ratio_extHyperlinks} \geq -1$
16	$-\texttt{ratio_intMedia} - \texttt{ratio_extMedia} \geq -100$
17	$\texttt{nb_hyperlinks} - \texttt{nb_extCSS} \geq 0$

Table 6: The 17 annotator-verified linear inequalities used as ground truth for the URL detection task.

fractions whose sum is at most one, while the corresponding media features sum to at most 100 (Constraints 15–16).

C.2 Detection Evaluation Protocol

Discovery–audit splits. For each dataset, we generate five 70/30 discovery–audit splits using different random seeds. Only the discovery partition is exposed to the constraint-discovery method; the audit partition and all ground-truth annotations remain hidden during discovery. Our agentic system examines two disjoint samples of 100 rows, perform three discovery runs and verifies them against the complete discovery partition, and may revise failed proposals for up to three rounds using verifier-provided counterexamples. Their naive baselines receive the same total sample budget of 200 rows in a single prompt and have no access to verification tools or iterative revision.

Contrastive detection tasks. Every example begins with an audit row that satisfies all ground-truth constraints and pairs it with a counterpart obtained by changing exactly one cell. For URL, we move a participating value across the target half-space boundary; each of the 17 inequalities receives 1,000 matched pairs. For NBA, we perturb a target-specific numerical column using noise scaled by the discovery-column standard deviation; each of the 12 equations also receives 1,000 pairs. We retain a mutation only when it violates the target constraint while satisfying every other ground-truth constraint. The URL and NBA tasks therefore contain 34,000 and 24,000 shuffled rows per split, respectively.

For ANXIETY, task allocation is proportional to each rule's full-data applicability support while preserving a budget of 1,950 matched pairs per split. Each of the 11 highest-anxiety implications applies to 651 of 11,000 source rows and receives 51 pairs. The occupation-to-group mapping applies to all 11,000 rows and receives 861 pairs, while the occupation-group-to-caffeine mapping applies to 6,751 rows and receives 528 pairs. A mutation changes only the dependent column to a disallowed category and is retained only when the target mapping fails and all other mappings still pass. Every constraint-specific task remains balanced between valid and invalid rows; support weighting captures how often a rule applies, not the natural prevalence of violations. Across the five splits, the three benchmarks contain 85,000, 60,000, and 9,750 matched pairs, respectively. Audit rows may be reused across constraint-specific tasks, so these totals count evaluation examples rather than unique source records.

Prediction and metric aggregation. We execute the constraints discovered by each method directly as an ensemble: a row is classified as invalid if any discovered constraint rejects it. We do not manually match discovered constraints to ground-truth formulas or revise them during evaluation. Malformed or non-executable constraints produce no violation predictions. We compute accuracy, precision, and recall for each detection task and take a support-weighted macro-average across the constraints of a given dataset. For metric m in a split, we report $\sum_c n_c m_c / \sum_c n_c$, where m_c is the metric for constraint c and n_c is its allocated number of matched audit pairs. This allocation is proportional to full-data applicability support for ANXIETY; the weights are equal for URL and NBA because every constraint receives 1,000 pairs. We report the mean and standard deviation of these weighted scores across the five discovery–audit splits. This protocol evaluates behavioral detection rather than exact formula recovery: a discovered constraint receives credit when it detects held-out violations even if its syntax differs from the corresponding curated rule.

D End-to-End Constraint-Aware Generation Details

D.1 Generator Training Details

Tables 7–10 report the resolved hyperparameter settings used for the four generators. Our CTGAN and TVAE implementations are adapted from the implementations released in the official code repository of Stoian et al. (2024).[11] Our TabDDPM implementation is adapted from the official implementation of Kotelnikov et al. (2023).[12] For Gaussian Copula, we use SDV 1.32.1's `GaussianCopulaSynthesizer`.[13]

[11] https://github.com/mihaela-stoian/ConstrainedDGM

[12] https://github.com/yandex-research/tab-ddpm

[13] https://docs.sdv.dev/sdv/modeling/single-table-synthesizers/gaussiancopulasynthesizer

CTGAN hyperparameter	Setting
Training epochs	300
Batch size	500
Noise/embedding dimension	128
Generator hidden layers	$[256, 256]$
Discriminator hidden layers	$[256, 256]$
Generator learning rate	2×10^{-4}
Discriminator learning rate	2×10^{-4}
Generator weight decay	1×10^{-6}
Discriminator weight decay	1×10^{-6}
PacGAN packing size (pac)	10
Optimizer	Adam
Gumbel–Softmax temperature	0.2

Table 7: CTGAN training configuration.

TVAE hyperparameter	Setting
Training epochs	300
Batch size	500
Latent/embedding dimension	128
Encoder/compression layers	$[128, 128]$
Decoder/decompression layers	$[128, 128]$
Optimizer	Adam
Learning rate	1×10^{-3}

Table 8: TVAE training configuration.

Gaussian Copula hyperparameter	Setting
Default numerical marginal distribution	Beta

Table 9: Gaussian Copula configuration.

TabDDPM hyperparameter	Setting
Optimizer updates	30,000
Maximum training batch size	4,096
Optimizer	AdamW
Initial learning rate	1×10^{-3}
Learning-rate schedule	Linear decay toward zero
Denoising MLP hidden layers	$[256, 256]$
Hidden-layer dropout	0
Time/label embedding dimension	128
Diffusion timesteps	1,000
Diffusion β schedule	Cosine
Numerical loss	MSE
EMA decay	0.999
FLIGHTS-only exception	
Hidden layers	$[256, 512, 512, 256]$
Diffusion timesteps	100

Table 10: TabDDPM training configuration, including the FLIGHTS-specific override.

D.2 Definition of Metrics

Constraint metrics. Constraint violation rate (CVR) and sample-wise constraint violation coverage (sCVC) are common constraint metrics (Stoian et al., 2025). Let n be the number of synthetic rows, $\mathcal{C}_f$ the constraints in family f, and $v_{ic} \in \{0, 1\}$ indicate whether row i violates constraint c. We define

$$\mathrm{CVR}_f = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[\sum_{c \in \mathcal{C}_f} v_{ic} > 0 \right].$$

$$\mathrm{sCVC}_f = \frac{1}{n|\mathcal{C}_f|} \sum_{i=1}^{n} \sum_{c \in \mathcal{C}_f} v_{ic}.$$

CVR is the fraction of rows that violate at least one constraint, while sCVC is the average fraction of constraints violated per row. Both metrics lie in $[0, 1]$, zero indicating no measured violations under the evaluated constraints. As the number of constraints grows, CVR can easily approach one because any single violation marks the entire row as violating; sCVC remains more informative in this regime because it measures violation density over all row–constraint checks.

For each evaluated equational constraint c over columns S_c, let the nonempty set $\mathcal{T}_c \subseteq S_c$ contain exactly the verified repair targets. A repair derives a target column $j \in \mathcal{T}_c$ from the remaining involved columns $S_c \setminus \{j\}$. We use the validated reconstruction function $g_{c \to j}$ implemented by the generated `fix(df)` code in the equational-constraint enforcement procedure described in Section 3.2. The evaluator considers every verified target direction and retains the best coefficient of determination. Equational consistency is

$$R^2_{\mathrm{eq}} = \frac{1}{|\mathcal{C}_{\mathrm{eq}}|} \sum_{c \in \mathcal{C}_{\mathrm{eq}}} \max_{j \in \mathcal{T}_c} R^2\big(x_j, g_{c \to j}(x_{S_c \setminus \{j\}})\big).$$

Higher values are better, and one indicates exact consistency with every evaluated equation.

Let $\mathcal{F}$ be the joint feasible region of the discovered linear constraints, and let s_j be the population standard deviation of numerical column j in the real training data. To keep the normalization defined for constant columns, set $\widetilde{s}_j = s_j$ when $s_j > 0$ and $\widetilde{s}_j = 1$ when $s_j = 0$, and let $\widetilde{D} = \mathrm{diag}((\widetilde{s}_j)_{j \in S_{\mathrm{num}}})$, where S_{num} indexes the numerical columns. Linear feasibility distance is

$$\mathrm{LFD} = \frac{1}{n} \sum_{i=1}^{n} \min_{z \in \mathcal{F}} \left\| \widetilde{D}^{-1}(x_i - z) \right\|_2.$$

Lower values are better, and zero indicates that every row lies in the joint feasible region.

Utility and univariate marginal fidelity. Under Train on Synthetic, Test on Real (TSTR) (Stoian et al., 2025), a predictor h_{syn} is fitted to a synthetic table and evaluated on the corresponding real test partition:

$$\text{TSTR} = \text{Score}\Big(h_{\text{syn}} = \text{Train}(\mathcal{D}_{\text{syn}}), \mathcal{D}_{\text{test}}^{\text{real}}\Big).$$

We report ROC-AUC for classification and R^2 for regression. Candidate classification models are logistic regression, decision tree, and XGBoost; candidate regression models are linear regression and XGBoost. For each dataset, we select the model with the highest mean Train on Real, Test on Real (TRTR) score. This selects logistic regression for HELOC; linear regression for FLIGHTS and NBA; and XGBoost for NEWS, STEEL, TAXI, and URL.

Column Shapes is the mean univariate similarity over the d columns. For a numerical column j, SDMetrics uses the KS complement; for a categorical column, it uses the total-variation complement:

$$q_j^{\text{num}} = 1 - \sup_z \left| \widehat{F}_{\text{real}}^j(z) - \widehat{F}_{\text{syn}}^j(z) \right|.$$

$$q_j^{\text{cat}} = 1 - \frac{1}{2} \sum_a \left| p_{\text{real}}^j(a) - p_{\text{syn}}^j(a) \right|.$$

$$\text{Column Shapes} = \frac{1}{d} \sum_{j=1}^{d} q_j.$$

All three scores lie between zero and one, with higher values indicating more similar marginal distributions.

D.3 Computational resources

Our experiments were conducted on a system equipped with two 64-core AMD EPYC 7662 processors (128 CPU cores in total) and four NVIDIA RTX 6000 Ada Generation GPUs, each with 48 GB of memory. The full design comprised 84 dataset–split–synthesizer settings (7 datasets $\times$ 3 splits $\times$ 4 synthesizers) and 252 synthetic-sample runs (three per setting). The recorded training and generation stages required approximately 18 hours of cumulative computation. If distributed well across all four GPUs, the workloads would require approximately 5 hours.

D.4 Full End-to-End Results

Table 11 reports Column Shapes for every dataset, generator, and output variant; higher values indicate greater univariate marginal fidelity to the real data.

		Column Shapes ↑		
Dataset	**Generator**	**Raw**	**Postprocessed**	**Δ**
FLIGHTS	CTGAN	0.906 (0.008)	0.912 (0.010)	+0.006
	Gaussian Copula	0.931 (0.001)	0.931 (0.001)	0.000
	TabDDPM	0.968 (0.004)	0.969 (0.003)	+0.001
	TVAE	0.932 (0.002)	0.935 (0.004)	+0.003
HELOC	CTGAN	0.906 (0.014)	0.900 (0.016)	-0.006
	Gaussian Copula	0.836 (0.021)	0.811 (0.033)	-0.025
	TabDDPM	0.850 (0.038)	0.850 (0.038)	0.000
	TVAE	0.930 (0.006)	0.927 (0.005)	-0.003
NBA	CTGAN	0.889 (0.019)	0.893 (0.012)	+0.004
	Gaussian Copula	0.824 (0.016)	0.846 (0.014)	+0.022
	TabDDPM	0.971 (0.009)	0.982 (0.001)	+0.011
	TVAE	0.896 (0.004)	0.897 (0.007)	+0.001
NEWS	CTGAN	0.862 (0.006)	0.854 (0.008)	-0.008
	Gaussian Copula	0.832 (0.005)	0.829 (0.003)	-0.003
	TabDDPM	0.544 (0.041)	0.541 (0.029)	-0.003
	TVAE	0.871 (0.009)	0.862 (0.010)	-0.009
STEEL	CTGAN	0.890 (0.015)	0.900 (0.012)	+0.010
	Gaussian Copula	0.816 (0.010)	0.861 (0.010)	+0.045
	TabDDPM	0.985 (0.001)	0.985 (0.001)	0.000
	TVAE	0.916 (0.032)	0.937 (0.016)	+0.021
TAXI	CTGAN	0.954 (0.004)	0.956 (0.005)	+0.002
	Gaussian Copula	0.856 (0.032)	0.863 (0.027)	+0.007
	TabDDPM	0.982 (0.012)	0.982 (0.012)	0.000
	TVAE	0.964 (0.002)	0.965 (0.002)	+0.001
URL	CTGAN	0.895 (0.019)	0.898 (0.018)	+0.003
	Gaussian Copula	0.862 (0.017)	0.861 (0.016)	-0.001
	TabDDPM	0.584 (0.011)	0.584 (0.013)	0.000
	TVAE	0.937 (0.001)	0.934 (0.004)	-0.003

Table 11: Column Shapes for all dataset–generator combinations before and after constraint postprocessing. Entries are means, with standard deviations in parentheses; each entry aggregates nine runs across three independent data splits. The final column gives Δ = Postprocessed − Raw for the displayed means; green and orange denote gains and losses, respectively. Higher is better.

Table 12 reports the complete utility results, including the selected dataset-level TRTR result and the raw and constraint-postprocessed TSTR score for every dataset–generator pair.

Table 13 reports every raw dataset–generator constraint result. We report only raw scores because postprocessing addresses the applicable constraints effectively and attains perfect constraint scores throughout: CVR, sCVC, and LFD are zero, while equational R^2 is one.

Dataset	Task / metric	Selected evaluation model	Generator	TRTR ↑	Raw TSTR ↑	Postprocessed TSTR ↑	Δ
FLIGHTS	Regression (R^2)	Linear regression	CTGAN	0.986 (0.000)	0.645 (0.037)	0.661 (0.040)	+0.016
			Gaussian Copula		0.963 (0.008)	0.964 (0.002)	+0.001
			TabDDPM		0.839 (0.052)	0.745 (0.111)	-0.094
			TVAE		0.875 (0.078)	0.874 (0.077)	-0.001
HELOC	Classification (ROC-AUC)	Logistic regression	CTGAN	0.802 (0.003)	0.720 (0.028)	0.720 (0.029)	0.000
			Gaussian Copula		0.775 (0.020)	0.784 (0.004)	+0.009
			TabDDPM		0.775 (0.009)	0.777 (0.008)	+0.002
			TVAE		0.790 (0.002)	0.790 (0.002)	0.000
NBA	Regression (R^2)	Linear regression	CTGAN	0.938 (0.014)	0.712 (0.043)	0.614 (0.092)	-0.098
			Gaussian Copula		0.923 (0.014)	0.892 (0.044)	-0.031
			TabDDPM		0.885 (0.011)	0.900 (0.008)	+0.015
			TVAE		0.902 (0.021)	0.845 (0.038)	-0.057
NEWS	Classification (ROC-AUC)	XGBoost	CTGAN	0.726 (0.002)	0.634 (0.021)	0.628 (0.021)	-0.006
			Gaussian Copula		0.589 (0.008)	0.588 (0.010)	-0.001
			TabDDPM		0.668 (0.003)	0.668 (0.006)	0.000
			TVAE		0.628 (0.014)	0.620 (0.014)	-0.008
STEEL	Regression (R^2)	XGBoost	CTGAN	0.999 (0.000)	0.880 (0.010)	0.996 (0.001)	+0.116
			Gaussian Copula		−0.850 (1.011)	0.914 (0.113)	+1.764
			TabDDPM		0.986 (0.001)	0.998 (0.000)	+0.012
			TVAE		0.930 (0.016)	0.997 (0.000)	+0.067
TAXI	Regression (R^2)	XGBoost	CTGAN	0.801 (0.004)	0.625 (0.040)	0.647 (0.033)	+0.022
			Gaussian Copula		0.633 (0.042)	0.629 (0.046)	-0.004
			TabDDPM		0.447 (0.184)	0.491 (0.050)	+0.044
			TVAE		0.729 (0.044)	0.738 (0.042)	+0.009
URL	Classification (ROC-AUC)	XGBoost	CTGAN	0.994 (0.000)	0.840 (0.034)	0.839 (0.019)	-0.001
			Gaussian Copula		0.903 (0.005)	0.907 (0.011)	+0.004
			TabDDPM		0.487 (0.095)	0.541 (0.057)	+0.054
			TVAE		0.966 (0.006)	0.965 (0.007)	-0.001

Table 12: Complete downstream utility results (mean; standard deviation in parentheses). TRTR aggregates three splits for the selected dataset-level model; TSTR aggregates nine runs across three splits for each dataset–generator pair. The final column reports Δ = Postprocessed − Raw for displayed TSTR means (green: gain; orange: loss). Regression uses R^2 and classification uses ROC-AUC; higher is better.

		Logical dependency		Equational			Linear		
Dataset	Generator	CVR ↓	sCVC ↓	CVR ↓	sCVC ↓	R^2 ↑	CVR ↓	sCVC ↓	LFD ↓
FLIGHTS	CTGAN	0.826 (0.021)	0.455 (0.013)	1.000 (0.000)	0.996 (0.000)	0.301 (0.031)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	Gaussian Copula	0.994 (0.002)	0.757 (0.029)	1.000 (0.000)	0.998 (0.000)	−0.214 (0.005)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	TabDDPM	0.116 (0.004)	0.057 (0.005)	1.000 (0.000)	0.982 (0.001)	0.691 (0.033)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
	TVAE	0.702 (0.004)	0.332 (0.002)	1.000 (0.000)	0.994 (0.001)	0.474 (0.010)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)
HELOC	CTGAN	0.013 (0.007)	0.013 (0.007)	–	–	–	0.715 (0.051)	0.095 (0.017)	0.698 (0.102)
	Gaussian Copula	0.125 (0.056)	0.125 (0.056)	–	–	–	0.804 (0.301)	0.114 (0.063)	2.254 (3.121)
	TabDDPM	0.003 (0.002)	0.003 (0.002)	–	–	–	0.310 (0.017)	0.050 (0.003)	1.462 (0.645)
	TVAE	0.020 (0.012)	0.020 (0.012)	–	–	–	0.519 (0.021)	0.055 (0.007)	0.261 (0.024)
NBA	CTGAN	–	–	1.000 (0.000)	0.955 (0.029)	−6.140 (10.488)	0.127 (0.220)	0.012 (0.020)	0.076 (0.132)
	Gaussian Copula	–	–	1.000 (0.000)	1.000 (0.000)	0.162 (0.438)	0.082 (0.141)	0.007 (0.013)	0.019 (0.033)
	TabDDPM	–	–	0.997 (0.001)	0.851 (0.005)	0.302 (0.310)	0.033 (0.057)	0.003 (0.005)	0.001 (0.002)
	TVAE	–	–	1.000 (0.000)	0.970 (0.007)	0.475 (0.139)	0.080 (0.138)	0.007 (0.013)	0.019 (0.033)
NEWS	CTGAN	0.186 (0.056)	0.019 (0.003)	1.000 (0.000)	0.950 (0.021)	−0.229 (0.052)	0.413 (0.578)	0.044 (0.062)	0.151 (0.210)
	Gaussian Copula	0.284 (0.057)	0.032 (0.004)	1.000 (0.000)	1.000 (0.000)	−0.222 (0.017)	0.455 (0.634)	0.047 (0.065)	0.088 (0.124)
	TabDDPM	0.040 (0.027)	0.004 (0.003)	1.000 (0.000)	0.529 (0.143)	−0.450 (0.398)	0.505 (0.451)	0.106 (0.105)	5.580 (5.907)
	TVAE	0.298 (0.041)	0.032 (0.008)	1.000 (0.000)	0.966 (0.031)	0.134 (0.015)	0.319 (0.449)	0.031 (0.044)	0.114 (0.159)
STEEL	CTGAN	0.032 (0.003)	0.032 (0.003)	0.993 (0.005)	0.669 (0.114)	0.647 (0.028)	0.007 (0.009)	0.000 (0.000)	0.003 (0.005)
	Gaussian Copula	0.383 (0.017)	0.383 (0.017)	0.999 (0.001)	0.797 (0.064)	0.359 (0.049)	0.213 (0.368)	0.009 (0.015)	0.074 (0.128)
	TabDDPM	0.000 (0.000)	0.000 (0.000)	0.949 (0.031)	0.503 (0.021)	0.951 (0.003)	0.003 (0.004)	0.000 (0.000)	0.000 (0.001)
	TVAE	0.054 (0.011)	0.054 (0.011)	0.986 (0.010)	0.579 (0.015)	0.639 (0.057)	0.007 (0.011)	0.000 (0.000)	0.002 (0.003)
TAXI	CTGAN	0.049 (0.004)	0.025 (0.002)	1.000 (0.000)	1.000 (0.000)	0.547 (0.039)	–	–	–
	Gaussian Copula	0.374 (0.086)	0.194 (0.045)	1.000 (0.000)	1.000 (0.000)	0.494 (0.237)	–	–	–
	TabDDPM	0.007 (0.003)	0.004 (0.001)	1.000 (0.000)	0.862 (0.012)	0.734 (0.292)	–	–	–
	TVAE	0.059 (0.015)	0.030 (0.007)	1.000 (0.000)	1.000 (0.000)	0.529 (0.111)	–	–	–
URL	CTGAN	0.033 (0.012)	0.017 (0.006)	–	–	–	0.962 (0.016)	0.134 (0.021)	0.873 (0.171)
	Gaussian Copula	0.025 (0.001)	0.012 (0.001)	–	–	–	0.852 (0.084)	0.080 (0.020)	0.211 (0.080)
	TabDDPM	0.020 (0.035)	0.010 (0.018)	–	–	–	0.691 (0.533)	0.183 (0.229)	6.320 (6.907)
	TVAE	0.009 (0.001)	0.005 (0.000)	–	–	–	0.606 (0.014)	0.047 (0.008)	0.242 (0.031)

Table 13: Raw constraint results for every dataset–generator combination. Entries are means, with standard deviations in parentheses, over nine runs spanning three independent data splits. CVR, sCVC, and LFD are lower-is-better metrics; equational R^2 is higher-is-better. Dashes indicate that a constraint family is unavailable for a dataset. News linear metrics are computed over the two splits containing detected linear constraints; the zero-constraint split is excluded ($n = 2$).

E Ablation Studies

E.1 Constraint-Discovery Hyperparameter Sensitivity

We examine three inference-time hyperparameters: discovery rounds R_{disc}, context rows n_{ctx}, and the maximum number of validation counterexamples returned per revision n_{cex}. Starting from $(R_{\text{disc}}, n_{\text{ctx}}, n_{\text{cex}}) = (3, 100, 20)$, we vary one parameter at a time, testing $R_{\text{disc}} \in \{1, 5\}$, $n_{\text{ctx}} \in \{50, 200\}$, and $n_{\text{cex}} \in \{0, 50\}$.

We evaluate the default and the six one-factor variants above—seven configurations in total—on three predefined splits of NEWS with GPT-5.6 Luna (OpenAI, 2026), yielding 21 discovery runs. Each run executes LD, equational, and linear discovery. We disable equational repair-function generation (`fix(df)`) so that token usage isolates constraint discovery.

For each run, *retained-constraint yield* is the number of hypotheses retained after full-table validation and family-specific consolidation, summed across the three families before enforcement-specific cross-family pruning. The count can therefore exceed its end-to-end counterpart. Because NEWS has no exhaustive gold constraint set, we treat higher yield as desirable: a larger validated and consolidated hypothesis set provides evidence of broader constraint coverage, rather than proof of exact semantic recovery. Figure 4 visualizes both yield and input-plus-output token usage, averaged over the three splits.

Among the tested round counts, three discovery rounds best balance yield and cost. Reducing R_{disc} from three to one cuts token use but lowers mean yield from 46.33 to 11.33. Increasing it to five more than doubles token usage while yielding fewer retained constraints (38.67), showing that additional rounds do not necessarily add unique hypotheses after validation and consolidation.

More context does not improve the cost–yield trade-off. Relative to the 100-row default, both 50 and 200 context rows consume more tokens while producing similar yields (48.00 and 44.33 versus 46.33). Thus, 100 rows is the most efficient context budget among those tested.

Counterexample feedback strongly improves retained-constraint yield. Removing it reduces mean yield from 46.33 to 19.33 without lowering token usage. Raising n_{cex} from 20 to 50 produces a similar yield (47.33) at greater cost.

Together, these results support $(3, 100, 20)$ as an efficient operating point in this setting; this one-factor-at-a-time study on one dataset and backbone does not establish a universally optimal configuration or statistical significance.

E.2 Equational Repair Order and Target Selection

To isolate the effect of equational repair order and target selection, we reuse the synthetic tables and discovered constraints from the end-to-end experiment in Section 4.2. We rerun only the equational-repair stage; LD and linear-inequality enforcement are disabled. The synthetic data, splits, generators, constraints, and validated target-specific repair functions are held fixed. The only intervention is the scheduling policy: our KS-complement-guided method (Appendix A.3) versus a structurally feasible random schedule.

Once an equation is repaired, all of its participating columns are frozen as future targets so that later reconstructions cannot invalidate it. Among all complete repair schedules satisfying this dependency rule, the KS-guided method first selects those with the highest minimum predicted change in KS complement and then chooses the schedule with the largest total predicted change. The random baseline instead randomly selects repair targets and their order, using backtracking whenever a choice prevents completion of the schedule.

The evaluation covers the five datasets containing discovered equational constraints—FLIGHTS, NBA, NEWS, TAXI, and STEEL—with four generators, three data splits, and three synthetic samples per split, yielding 36 matched inputs per dataset and 180 in total. Scheduling becomes consequential when equations share columns or admit multiple validated target-specific repairs, because each resolved equation removes columns from later target choices. We evaluate univariate marginal fidelity using Column Shapes and constraint satisfaction using equational CVR. Table 14 reports dataset-level results aggregated across generators, splits, and synthetic samples.

KS guidance yields higher marginal fidelity while both schedulers achieve exact equation satisfaction. Both feasible schedules produce zero equational CVR on all 180 matched inputs, so their difference lies in fidelity rather than constraint satisfaction. Relative to random feasible scheduling, KS guidance improves mean paired Column Shapes on four datasets by 0.020–0.053, with the largest gain on FLIGHTS, and is effectively tied on NEWS

	Column Shapes ↑		
Dataset	KS-guided	Random feasible	KS gain
FLIGHTS	0.878(0.037)	0.825(0.048)	**+0.053(0.020)**
NBA	0.916(0.026)	0.897(0.040)	**+0.020(0.024)**
NEWS	0.819(0.102)	0.818(0.101)	0.000(0.001)
TAXI	0.898(0.043)	0.875(0.051)	**+0.023(0.015)**
STEEL	0.890(0.046)	0.863(0.060)	**+0.028(0.031)**

Table 14: Equational-repair scheduling ablation, aggregated by dataset over four generators, three data splits, and three synthetic samples per split (36 matched inputs per dataset; 180 total). Values are means with sample standard deviations in parentheses. KS gain is computed per matched input as KS-guided minus random feasible, so positive values favor KS guidance. Equational CVR is zero under both schedules on every input and is therefore omitted.

at the reported precision (0.000). The equal-weight mean improvement across datasets is 0.025. Thus, in these settings, KS guidance yields higher univariate marginal fidelity than random feasible selection, without claiming global optimality.

E.3 Equational–Linear Repair Order

We next ablate the order of the two numerical repair stages. Logical dependencies involve only categorical columns, so their participating columns are disjoint from those involved in equational and linear repair. Consequently, LD repair cannot interfere with either numerical constraint family, and we apply it first in both conditions. We then vary the numerical order: *equational then linear* ($E \to L$), our default, versus *linear then equational* ($L \to E$).

This order matters because equations and linear inequalities can share numerical columns. Under $E \to L$, equational repair first establishes the equations; the subsequent projection then protects equation-participating columns and adjusts only the remaining mutable columns, thereby preserving the repaired equations while enforcing the inequalities. Under $L \to E$, no equations have yet been repaired, so the projection has no equation-derived protected columns and may modify any participating numerical column. Subsequent equational repair can then change shared columns again and move previously projected rows outside the linear feasible region.

We focus on NBA because its mined equational and linear constraints share participating columns. We reuse the trained generators, mined constraints, and corresponding synthetic tables from the end-to-end experiment. Both conditions start from the same synthetic table and hold the discovered constraints and family-specific repair routines fixed; the only intervention is the order of the two numerical stages. The ablation spans three data splits, four generators, and three independent synthetic draws per split, yielding 36 matched generator–split–draw settings (nine per generator). Table 15 reports the final linear CVR and LFD, aggregated over the nine settings for each generator. We also verify the final equational CVR under both orders to detect cross-family regressions.

Generator	Metric	$E \to L$	$L \to E$
CTGAN	CVR (%) ↓	0 (0)	7.331 (5.919)
	LFD ↓	0 (0)	1.583 (2.102)
TVAE	CVR (%) ↓	0 (0)	0.782 (1.193)
	LFD ↓	0 (0)	0.640 (1.225)
Gaussian Copula	CVR (%) ↓	0 (0)	9.919 (3.753)
	LFD ↓	0 (0)	1.219 (0.755)
TabDDPM	CVR (%) ↓	0 (0)	0.262 (0.366)
	LFD ↓	0 (0)	0.012 (0.014)
All	CVR (%) ↓	**0 (0)**	**4.574 (5.419)**
	LFD ↓	**0 (0)**	**0.863 (1.360)**

Table 15: Repair-order ablation on NBA. $E \to L$ is categorical → equational → linear repair; $L \to E$ swaps the two numerical stages. The **All** row aggregates all 36 matched settings. Entries are means, with sample standard deviations in parentheses. Final equational CVR is zero under both orders and is therefore omitted.

Only $E \to L$ achieves joint satisfaction in all 36 matched settings. Its final linear CVR and LFD are both zero, while the equations remain exactly satisfied. In contrast, $L \to E$ ends with linear violations in 24 of 36 settings (66.7%), raising aggregate CVR to 4.574% and LFD to 0.863. The reversed order yields nonzero mean CVR for every generator, led by Gaussian Copula (9.919%) and CTGAN (7.331%). Because both orders retain zero equational CVR, it loses linear feasibility rather than trading satisfaction between the two families.

Repair interference is directional. Under $L \to E$, equational reconstruction changes shared columns after projection and can move rows outside the linear feasible region. Under $E \to L$, the final projection protects equation-participating columns and, in these settings, restores linear feasibility without undoing the equations. Thus, projection last is the safer order when the two families overlap. Because NBA was selected specifically for such overlap, this ablation tests that interference mechanism rather than claiming that order matters when the families use disjoint columns.